2/93
SONDERHEFT

AMIGA MAGAZIN

Markt&Technik

DM 9,80
öS 79,-
sfr 9,80

AMIGA PROGRAMMIEREN

Disketten zum Heft als Public Domain. Mit allen Listings und vielen Extras

Faszination Programmieren

- ARexx von A bis Z
- Raytracing-Verfahren
- Alle Geheimnisse des Amiga 1200

Und mit vielen Knobeleien, Tips & Tricks, großem Programmierprojekt, tollem Wettbewerb u.v.m...

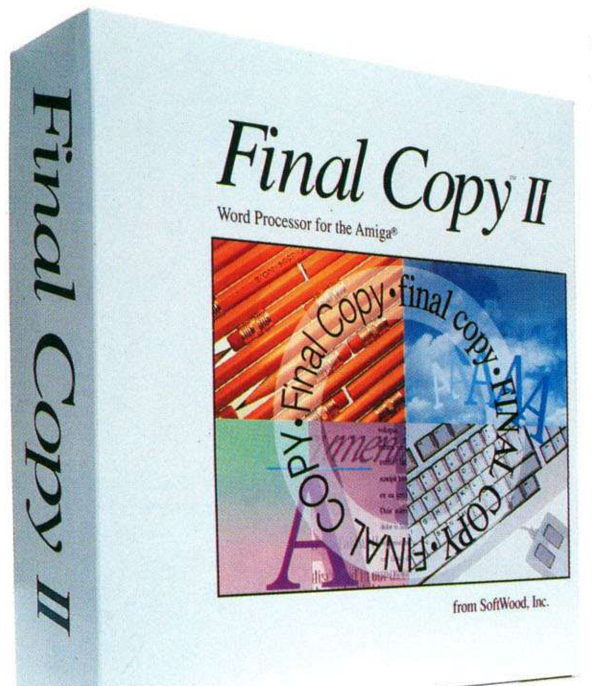
DAS LETZTE WORT

...in Sachen Textverarbeitung mit perfektem Ausdruck

Ein brandneues Textverarbeitungspaket – einfach das Muss für den, der höchste Ansprüche an seinen Amiga stellt: komplett in Deutsch und WYSIWYG!

Final Copy II ist nicht nur das derzeit leistungsfähigste Textverarbeitungspaket für den Amiga – mit allen Funktionen, die Sie von einer Textverarbeitungsoftware erwarten können – sondern es wartet auch noch mit einer Reihe weiterer DTP–Funktionen auf, die sonst nur in professionellen DTP–Systemen wie z.B. auf dem Macintosh zu finden sind. Es sind sogar voll skalierbare outline-Fonts auf allen Amigamodellen möglich - auch mit Kickstart 1.3.

Sie können mit Final Copy II die höchstmögliche Druckerauflösung in PostScript–Qualität erreichen, egal welchen Drucker sie benutzen. Sogar mit einem einfachen 9–Nadeldrucker ist das Ergebnis verblüffend.



Vielfältige und zeitungsgleiche Spalten- und integrierte Zeichenfunktionen für Rechtecke, Pfeile und Linien in jedem Winkel, Ellipsen etc., sowie farbigen Text und andere Formatierungsfunktionen, lassen Ihr Dokument so aussehen, wie Sie es sich vorstellen.

Final Copy II beinhaltet ein erweiterbares Wörterbuch mit über 142.000 Eintragungen, um Rechtschreibfehler automatisch auszuschließen. Weiterhin ist ein Synonymwörterbuch mit 580.000 Eintragungen inbegriffen.

Final Copy II ist einfach zu erlernen und anzuwenden. Sollten Sie trotzdem Unterstützung zu irgend einem Problem benötigen, leistet unsere Support hotline jedem registrierten Kunden der deutschen Version volle Unterstützung.



Ohne...



...und mit...



...Final Copy II

empf. VK-Preis: 299.– DM
im gut sortierten Fachhandel

Kompatibel mit:

Amiga®–A500/500+/600/600HD/1200/2000/2500/3000/4000 und jedem Workbench™–unterstützten Farb– und S/W–Grafik–Drucker.
Systemvoraussetzung: min. 1Mb. RAM und zwei Diskettenlaufwerke oder eine Festplatte [A600HD benötigt min. 1,5Mb.]. WorkBench 1.3/2.x.

Weitere Funktionen von Final Copy II:

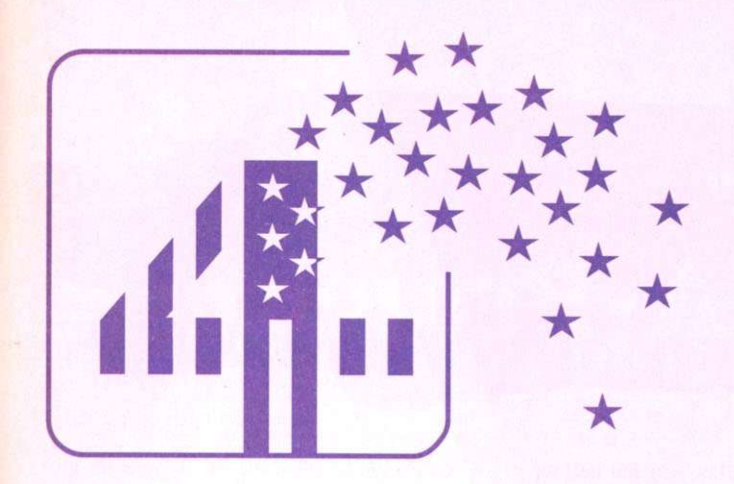
• Outline Fonts in allen Auflösungen von 4 bis 300 Punkt • Wählbare Druckqualität incl. PostScript®– Ausgabe und max. 4096 Farben • ARexx– Schnittstelle incl. Programm–Macros • Serienbriefe • Dokumentstatistik • Addition von Zahlenspalten • Text über Grafik • Automatischer Textfluß um Grafiken • Farbiger Text • Links, rechts, mitte und dezimale Tabulatoren • Absatzorientierung • Speicherbare Absatzformate • Importieren, Skalieren und Schneiden von IFF–, HAM– und 24Bit ILBM–Bilder • Ausrichtung an Hilfslinien • Einfügen und kopieren von horizontalen und vertikalen Linealen • Maße in Pica, Zoll und Millimeter • Frei definierbarer Zeilenabstand • Kapitälchen • Hoch– und Tiefstellen • Durchstreichen, einfach und doppelt unterstreichen • Darstellungsverkleinerung/–vergrößerung von 25% bis 400% bei freier Bearbeitung • Suchen und Ersetzen • Kopieren, Ausschneiden und Einfügen • Clipboard–Unterstützung • Einfügen von Systemzeit und/oder –datum sowie automatisch durchnumerierter Seiten • Frei definierbare Seitengröße • Layout– und Titelseiten • Rechte/linke Seite • Gehe zu Seite oder Einfügepunkt • Seiten– und Spaltenumbruch einstellbar • Unterstützung von großen Monitoren • Deutsche Silbentrennung.

Händlerverkauf durch:

H.S.&Y., ADX, Leisuresoft, Profisoft, Casablanca, GTI oder direkt bei:



AMIGAOBERLAND
IN DER SCHNEITHOHL 5 · D-6242 KRONBERG 2
TEL.: 06173/65001 · FAX: 06173/63385



Jetzt funkt´s im Karton

Dieses Sonderheft zeigt klar, daß der Amiga im Reigen der »Personal Computer« ganz vorne mitspielt. Wenn man ihn richtig programmiert, ist mehr aus ihm rauszuholen als aus jedem anderen System, egal ob von »Daktari«, »Mec« oder irgendeinem »ET« die Rede ist.

Allein was mit ARexx machbar ist, kann kein anderer PC derzeit bieten. ARexx und die Programmierung des neuen Amiga 1200 bilden daher zwei Schwerpunkte dieser Ausgabe »Faszination Programmieren«. Hier erfahren Sie, wie Sie z.B. mit ARexx 3-D-Grafik programmieren, was man mit dem 68020er-Prozessor im Amiga 1200 alles machen kann, und wie Sie die Grafikchips der neuen Amigas ausreizen.

Wir widmen uns aber nicht nur der neuen Amiga-Standard-Programmiersprache ARexx und dem Amiga 1200; auch ans neue OS 3.0 haben wir gedacht. So ist für alle neuen Amiga-Besitzer etwas dabei.

Schade, daß Commodore immer noch viel zu wenig tut, um den professionellen Character des Amigas zu stützen. Im Gegenteil – man scheint sich eher im Spiele-Sektor umzuschauen, wie die neuen Produktvorstellungen zeigen. Doch wir denken anders und regen z.B. die Entwicklung professioneller Software für den Amiga an. Wer die Ausgabe 1 von »Faszination Programmieren« bereits gelesen hat, ist sicher auf die Fortsetzung unseres großen Programmierprojekts gespannt. Mehr erfahren Sie auf der nächsten Seite.

An dieser Stelle Dank an alle, die sich an dem Projekt beteiligt haben, und auch an die, die sich noch beteiligen werden. Dank auch allen, die an dieser Ausgabe mitgearbeitet haben: Wie bei der Nummer 1 haben sich alte Amiga-Hasen kräftig ins Zeug gelegt, um dieses Sonderheft so rund wie möglich zu machen..

Viel Spaß mit der zweiten Ausgabe von »Faszination Programmieren« wünscht Ihr AMIGA-Team:

Rainer Zeitler & Ulli Brieden

Fortsetzung: Programmierwettbewerb

Das AMIGA-Projekt

Im ersten Sonderheft »Faszination Programmieren« riefen wir alle enthusiastischen Amiga-Programmierer auf, bei unserem Wettbewerb mitzumachen. Die Resonanz war riesig.

von Rainer Zeitler

Die Idee gefiel allen: Im Amiga-Softwaresektor herrscht chronischer Mangel an professionellen Programmen. Die Redaktion von »Faszination Programmieren« nahm das zum Anlaß, einen Wettbewerb ins Leben zu rufen und mit Hilfe der Leser ein umfangreiches Projekt durchzuführen. Zwischen drei Vorschlägen galt es auszuwählen:

- Tabellenkalkulation
- Textverarbeitung
- Projekt-Management für verschiedene Programmiersprachen

Die Vorschläge und Anregungen unserer Leser, die im Laufe der Zeit die Redaktion erreichten, übertrafen weit unsere Erwartungen. Seitenlange Abhandlungen über die in Aussicht gestellten Projekte, detaillierte Vorgehensweisen über die Durchführung etc. Hierfür möchten wir allen Einsendern herzlichst danken, die sich aus unterschiedlichsten Berufen zusammensetzten: Studenten, Berufsprogrammierer, Schüler und Hobby-Programmierer.

Projekt-Management liegt vorn

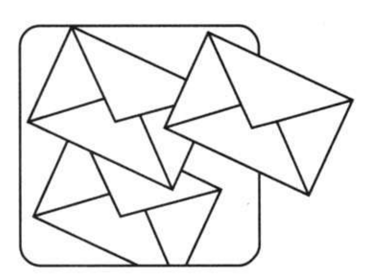
Überraschend war das Ergebnis der Auswertung. Die Redaktion rechnete damit, daß sich die Projekte »Tabellenkalkulation« bzw. »Textverarbeitung« ein Kopf-an-Kopf-Rennen liefern würden – weit gefehlt: ca. 80 Prozent entschieden sich fürs Projekt-Management. Hier einige Leserstimmen:

Unter den von Ihnen vorgeschlagenen Projekten reizt mich besonders das letztere (Projekt-Management für verschiedene Programmiersprachen), da dies meiner Meinung nach die größte Herausforderung darstellt und ich mir durchaus vorstellen kann, daß gerade dafür ein enormer Bedarf besteht. Ich stelle mir so eine Art »Programmers Workbench« vor, u.a. mit einem »Source-Revision-Control-System«.

Jürgen Zimmermann, Altleiningen

Als ich Ihren Artikel las, war ich hell begeistert. Ich würde mich gerne als Programmierer zur Verfügung stellen. Es soll eine Applikation entstehen, die im Prinzip die Zusammenarbeit verschiedenener Programmierer an einem Projekt erleichtern soll. Ich habe mir schon lange die Realisierung so eines Projekts vorgenommen – für einen einzelnen Programmierer ist aber dieser Aufwand kaum zu tragen.

Jonas Greutert, Maur (Schweiz)



Zuerst möchte ich Ihnen zum ersten Programmier-Sonderheft beglückwünschen, sein Informationsgehalt ist äußerst wertvoll. Die Idee mit dem Softwareprojekt ist einfach genial. Bei der Durchführung sollten Sie eine möglichst hohe Leserbeteiligung anstreben und den Leser stets über den aktuellen Stand informieren. Auf jeden Fall würde es mich reizen, einmal an einem größeren Softwareprojekt teilnehmen zu dürfen.

Michael Haag, Karlsruhe

Ich würde mich gerne an Ihrem Amiga-Programmierprojekt beteiligen. Warum ich bereit bin, auch noch Freizeit zum Programmieren zu opfern? Nun, die kommerzielle Welt ist etwas trist (Cobol und C) für jemanden, der in seinem Studium mit etwas besseren Ideen konfrontiert wurde. Außerdem denke ich, daß es einer der Vorzüge von Modula-2 oder Oberon ist, mit geringem Zeitaufwand gute Ergebnisse zu erzielen. Mein Themenvorschlag: Natürlich das Projekt-Management, das ich für sehr sinnvoll halte.

Gernot Daum, München

Eure Idee, ein großes Programmierprojekt ins Leben zu rufen, finde ich echt gut und würde mich daher gerne daran beteiligen. Obwohl mir noch nicht ganz klar ist, wie ein Projekt-Manager aussehen sollte, reizt mich diese Variante am meisten, da es so etwas für den Amiga noch nicht gibt.

Holger Rabbach, Solingen

Das Ziel steht also fest. Nun gehts an die Umsetzung. Für die Planung benötigen wir Ideen, welche Aufgaben der Applikation zukommen, welche Features implementiert werden müssen etc.? Das Projekt-Management soll, von Programmiersprachen unabhängig, der Entwicklung von umfangreichen Anwendungen dienen. Der Zugang zu Compilern oder Assemblern geschieht über eine definierte Schnittstelle, z.B. ARexx oder Aufruf über eine eigene DOS-Shell. Hier einige Anregungen:

- Integrierter Editor mit kontextorientierter Ausgabe. Abhängig von der eingestellten Programmiersprache sollten Schlüsselwörter, Bezeichner etc. besonders gekennzeichnet werden. Außerdem muß die Möglichkeit gegeben sein, auf Tastendruck z.B. die zugrundeliegende Struktur für eine Variable anzuzeigen. Selbstredend, daß Folding-Technik (Verstecken ganzer Funktionen) implementiert sein muß. Die Suchfunktion muß sich optional auf alle Module erstrecken. Ein Mausklick auf einen Funktionsnamen sollte diese anzeigen, unabhängig davon, ob die Funktion im aktuellen oder einem anderen Modul existiert.
- Modulverwaltung mit der automatischen Vergabe von Versionsnummern, Make-Funktion, Auffinden von Referenzen und Abhängigkeiten innerhalb der Programmmodule.
- Generierung von Ablaufdiagrammen und erzeugen von Quelltext.
- Oberflächengenerator fürs mausorientierte Anlegen von Fenstern, Menüs etc.

Gute Ideen sind gefragt

Also: Was soll unser – bzw. Ihr – Programm können? Schreiben Sie Ihre Ideen auf und schicken Sie sie uns. Grundlage für das Programm ist OS 2.0 und höher. Schicken Sie Ihre Vorschläge bis zum 30. November 1993 an:

AMIGA-Redaktion
»Faszination Programmieren«
Markt & Technik Verlag AG
Postfach 1304, 85531 Haar bei München

Die Programmierer, die ihr Interesse an unserem Projekt bekundet haben, werden wir in Kürze benachrichtigen. Abschließend der Gewinner der 25-Mhz-GVP-MC68030-Turbokarte, gestiftet von der Firma DTM Computersysteme. Gewonnen hat

Peter Gerlach aus Mettmann

Herzlichen Glückwunsch. In der nächsten Runde gibt´s dann wieder einen tollen Preis zu gewinnen, vielleicht sogar einen neuen Amiga (mehr wird nicht verraten). ■

INHALT

Wissenswertes

Das unbekannte V

Seit fast einem Jahr gibt es den Amiga 4000 und den Amiga 1200 mit dem neuen AA-Chipset. Zeit genug, um den Nachfolger des legendären Amiga 500 genauer kennenzulernen. Wir zeigen Ihnen, was Sie in Sachen Programmierung und Bedienung des Amiga 1200 alles wissen sollten, um die volle Leistung des doch so anderen Amiga auch richtig auszunutzen.

von Alexander Kochann und Oliver Reiff

Als der Amiga 1200 vor rund einem Jahr vorgestellt wurde, sorgte er mit seinem niedrigen Preis und seinen neuen, fantastischen Fähigkeiten im Bereich Grafik gleich für Aufsehen. Und daß sich der »kleine Bruder« des Amiga 4000 mittlerweile auf dem Markt etabliert hat, beweisen seine Verkaufszahlen: 45 000 gingen allein im ersten Quartal 1993 über die Ladentische.

Dadurch ist auch für Programmierer eine neue Basis geschaffen worden, die sie nicht außer acht lassen sollten. Allerdings muß man einiges über den Neuen wissen, wenn man etwas aus ihm herauskitzeln möchte.

Neue Features für Ihre Programme

Am auffälligsten ist wohl das AA-Chipset. Hier in aller Kürze die wichtigsten Daten:

In allen Auflösungen stehen 256 aus einer Palette von 16 777 216 Farben zur Verfügung. Dazu kommen die Modi HAM6 und HAM8, in denen mit kleinen Einschränkungen – im Prinzip – alle Farben angezeigt werden können. In der Praxis kann man das jedoch nie erreichen; selbst auf einem Amiga 4000 steht nicht genügend Chip-RAM zur Verfügung, um einen Screen mit ebensovielen Pixeln zu öffnen. Wegen der etwas umständlichen Handhabung der HAM-Modi werden diese nur von Grafikprogrammen verwendet. Andere Anwendungen und Spiele verwenden in der Regel maximal 256 Farben.

Auch bei den Auflösungen hat sich einiges getan. So bietet der PAL-SuperHiRes-Modus in der Interlace-Darstellung mit 1448 x 566 Pixeln die höchste horizontale Auflösung. Der DblPAL-HiRes-Interlaced dagegen schafft die höchste vertikale Auflösung mit 720 x 1128 Bildpunkten. Diese extremen Auflösungen werden jedoch allein schon wegen des berüchtigten Interlace-Flimmerns in der Praxis fast nicht genutzt. Ein sog. ScanDoubler könnte dies zwar beheben, hat aber bei hohen horizontalen Auflösungen (SuperHiRes) Probleme mit der Darstellung, da jede zweite Spalte verschluckt wird oder der Modus diese erst gar nicht darstellt.

Wer noch mit einem Fernseher oder einem alten Monitor, wie dem 1084S arbeitet, hat sowieso nur die Wahl zwischen PAL und NTSC. Um die neuen VGA-Modi wie DoublePAL benutzen zu können, empfiehlt sich der Anschluß eines MultiSync-Monitors. Wer einen reinen VGA-Monitor am Amiga 1200 betreiben möchte, sollte sich einen ScanDoubler zulegen, damit er sich auch die viel benutzten Modi PAL und NTSC anzeigen lassen kann. Andernfalls kann man nicht einmal das Bootmenü erkennen, da es nur in diesen Modi arbeitet.

Im Gegensatz zu den neuen Grafikchips Lisa und Alice blieb der Soundchip unangetastet. Paula verwaltet weiterhin die vier Stereo-Tonkanäle mit acht Bit. Neuerungen wie DSP sind jedoch für die Zukunft zu erwarten.

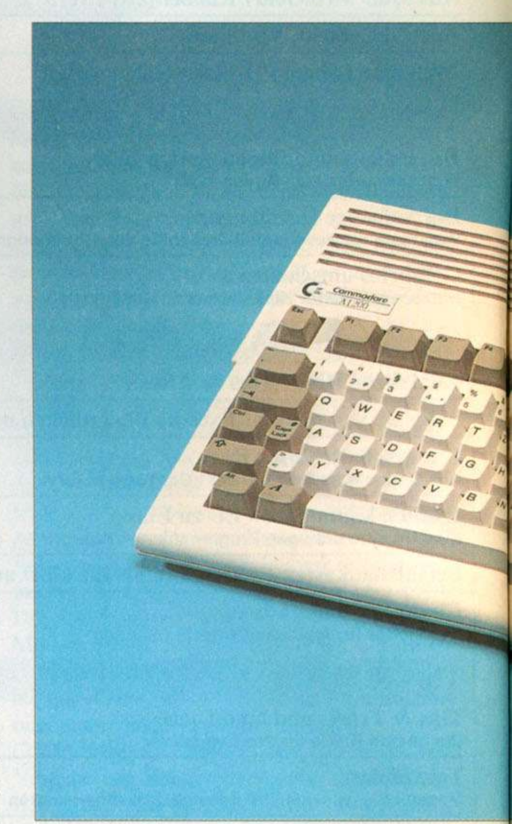
Auch der mit 14 MHz getaktete 68020-Prozessor trägt seinen Teil zur verbesserten Leistungsfähigkeit bei. Dadurch wird die reine Rechenleistung etwa verdoppelt. Durch geschickte Nutzung des erweiterten Befehlssatzes und der komplexen Adressierungsarten lassen sich Programme zusätzlich beschleunigen. Diese laufen dann natürlich nur noch ab dem Amiga 1200 »aufwärts«. Da bei der Arbeit mit dem AA-Chipset mindestens ein 68020er vorausgesetzt werden kann, sollten dessen Möglichkeiten auch genutzt werden. Durch den internen Erweiterungsschacht kann aber ohne Probleme eine Turbokarte mit einem noch leistungsfähigeren Prozessor wie dem 68040er eingesetzt werden.

Außerdem läßt sich der Speicher dort mit 32-Bit-RAM erweitern. Zu den 2 MB Chip-RAM kommen dann bis zu 4 MByte Fast-RAM, wenn man keinen höheren Prozessor einbaut. Nicht ganz so schnell aber wesentlich einfacher



```
AttnFlags   equ 296
AFB_68020   equ 1
   move.l   4.w,a6
   move.w   AttnFlags(a6),d0
   btst     #AFB_68020,d0
   bne      ProzessorOk
*** Fehlerroutine und Ende ***     ...     ...
ProzessorOk
   MC68020         * nicht DevPac
*** 68020 Befehle nun möglich ***
   ...      ...         © 1993 M&T
```

Listing 1: Hiermit testen Sie, ob im Amiga ein MC68020 steckt



»Amiga 1200«: Der Neue von Commodor grammieren«, was Sie beim Programmier

zu handhaben sind die mit 16 Bit Breite arbeitenden Speicherkarten für den PCMCIA-Slot. Einstecken und loslegen heißt hier das Motto.

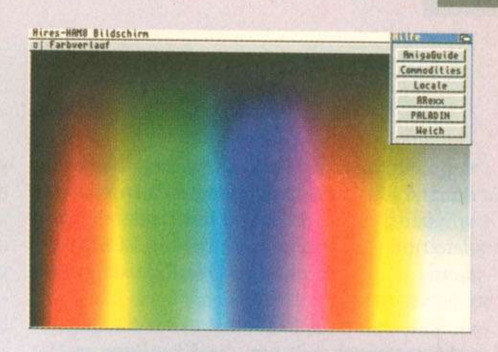
Neben der neuen Hardware wurde auch die Software erheblich erweitert. Umsteigern von Kickstart 1.2/1.3 wird gleich das professionelle Design (kleines Bild rechts oben) auffallen, während OS2.0-Benutzer kaum optische Veränderungen feststellen können. Für Individualisten stehen statt einem Preferences-Programm jetzt gleich eine ganze Schublade davon zur Verfügung. Neben Font, Farben und Screentyp kann man seit OS2.1 auch die Sprache einstellen, die der Amiga verwenden soll. So erscheint die Workbench jetzt in Deutsch oder Schwedisch, ganz wie man will...

Des einen Freud...

So angenehm dem Benutzer die Vielfalt an Voreinstellungen auch auffällt, für viele Programmierer ist sie ein Dorn im Auge, denn diese Individualität erfordert natürlich einen erhöhten Programmieraufwand.

Bereits beim Programmstart sollten grundlegende Voraussetzungen geprüft werden. Dazu zählen u.a. das verwendete Betriebssystem,





n Amiga 1200

Vesen Amiga 1200



**ch viel Neues zu bieten. Wir zeigen Ihnen in dieser Ausgabe von »Faszination Pro-
hten müssen und was Sie alles aus dem 1200er rausholen können.**

vorhandene Prozessoren und Grafikchips. Bedenken Sie dabei auch, daß auch neuere Betriebssystemversionen noch auf »Minimalsystemen« (also MC68000, altes Chipset und 512 KByte Chip-RAM) laufen können. Am wichtigsten ist wohl die verwendete CPU. Diese ist ebenso wie die Matheprozessoren und die MMU in den AttnFlags der ExecBase notiert. Erst nach einem solchen Test (Listing 1), sollten Sie Ihren Assembler in einen höheren 680x0-Modus schalten, um auch die erweiterten Befehle voll nutzen zu können.

Wie die Prozessoren in ExecBase finden Sie in GfxBase die Bits für die Grafikchips . Dort enthält das UBYTE gb_ChipRevBits0 alles Wissenswerte über das verwendete Chipset. In der Regel sind die Informationen für normale Anwenderprogramme jedoch uninteressant, da sie nur an spezifischen Möglichkeiten des Chipsets interessiert sind. Diese können über das Betriebssystem differenzierter abgefragt werden. Es ist sogar aufwärtskompatibel, wenn sich Programme direkt nach den Daten des Systems richten und nicht nach festen Werten. Denn noch höhere Auflösungen und mehr Farben werden in Zukunft möglich sein oder sind bereits mit Grafikkarten verfügbar.

Programme, die erst ab dem AA-Chipset arbeiten können, weil sie z.B. Interleave-Bitmaps oder eigene Copperlisten benötigen, sollten diese Bits aber unbedingt testen und eine entsprechende Fehlermeldung ausgeben, damit der Benutzer weiß, woran er ist und nicht ewig nach der Absturzursache suchen muß.

Jetzt sollten alle benötigten System-Ressourcen wie Libraries, Devices und Speicher besorgt werden. Danach kann man sich an das Öffnen eines Fensters wagen. Dies ist gar nicht so einfach, will man alle Eventualitäten berücksichtigen. Flexibilität ist also gefragt. Nicht nur Höhe, Breite und Farbtiefe der Workbench können variieren. Auch Font und DrawInfo des Screens sind zu berücksichtigen. Gerade die DrawInfo-Struktur bietet viele Möglichkeiten. Dort lassen sich vor allem die vom System verwendeten Farben auslesen. So wird Text nicht immer mit der Farbe Nummer 1 ausgegeben, die 3-D-Kanten und Titelbalken nicht immer mit den gleichen Farben dargestellt. Diese variieren vielmehr von Screen zu Screen.

Öffnet man das Fenster auf einem eigenen Screen, verlagert man das Problem nur. Zwar kann man dann alle verwendeten Fonts, Pens und Farben selbst festlegen, aber erstens geht das an der Individualität des Benutzers vorbei und außerdem muß man dann einen eigenen Bildschirm öffnen. Ob die benutzten Grafikchips diesen Bildschirm überhaupt darstellen können, erkennen Sie am Rückgabewert von »ModeNotAvailable()« der »graphics.library« und nicht erst daran, ob »OpenScreen()«, oder besser »OpenScreenTagList()«, fehlschlägt. Denn selbst bei diesen Routinen kann es zu Abstürzen kommen, wenn der Screen gar nicht dargestellt werden kann. Ein weiteres Problem ergibt sich aus dem vom Benutzer verwendeten Monitor. Denn bekanntlich kann z.B. der 1084S ja keine DblPAL-Modi darstellen. Deshalb empfiehlt es sich, den Benutzer bei der Verwendung des Screens ein Wörtchen mitreden zu lassen. Am besten mit einem Screen-Mode-Requester der »asl.library«.

Apropos Requester: Natürlich sollten auch Zeichensätze und Dateien über die Requester der »asl.library« auswählbar sein. Es ist immer wieder ärgerlich, wenn man den ganzen Pfad einer Datei mit eintippen muß oder den Text nicht erkennt, weil der Font zu klein ist (z.B. Topaz 8 bei den hohen Auflösungen von Grafikkarten, zukünftigen Auflösungen oder sogar DblPAL-Interlace).

Auch eigene Requester, an die sich der Benutzer für jedes Programm neu gewöhnen muß, sollten eigentlich der Vergangenheit angehören, da die ASL-Requester wirklich universell einsetzbar sind. Zwar waren die ersten Versionen des ASL-Filerequesters ziemlich langsam, inzwischen sind sie aber beim Einlesen von Verzeichnissen schneller als mancher selbstgestrickter, dazu sind sie komfortabler und locale-fähig.

Spiel läuft nicht – Geld zurück

Überhaupt sollten Sie so viel wie möglich dem Betriebssystem überlassen. Dies erspart Zeitaufwand beim Programmieren, erleichtert die Bedienung und ist garantiert ziemlich aufwärtskompatibel. Bestes Beispiel dafür ist die Funktion »GetCC()« der »exec.library«. Diese ersetzt den Befehl »move sr,d0«, was nur auf dem MC68000 im User-Modus erlaubt ist. Auf allen anderen Prozessoren führt der Befehl zum Absturz wegen Privilegverletzung. Das war natürlich nicht vorhersehbar, als nur der MC68000 auf dem Markt war.

Kompatibilität ist sowieso zu einem Reizwort für die Amiga-Anwender geworden. Neulich wollten ein Junge sogar seinen Amiga 1200 an den Händler zurückgeben, weil dieser kaputt sein müsse. Die Hälfte seiner alten Spiele laufe nicht. Und das waren bestimmt nicht nur die Spiele, die seinem alten Amiga 500 auf

den Leib, bzw. die Hardware, geschrieben wurden. Auch ernstzunehmende Anwendungen wie beispielsweise BeckerText hatten ein Update nötig, da sie mit dem neuen Betriebssystem nicht zurechtkamen. Wie läßt sich so etwas vermeiden?

Nun, daß man ROM-Adressen niemals direkt anspringt und nur die ExecBase an der Adresse $00000004 feststeht, sollte inzwischen hinreichend bekannt sein. Natürlich ist auch auf das Multitasking Rücksicht zu üben. Dies heißt jedoch nicht nur, daß man sich Speicher vom System besorgen läßt und ihn ebenso wieder freigibt. Multitasking heißt auch, die vorhandenen Resources so gut wie möglich zu nutzen und gerade mit dem Speicherverbrauch sparsam umzugehen.

Fehler beim Umstieg vermeiden

Das bedeutet, daß man die vorhandenen Möglichkeiten des Betriebssystems so gut wie möglich ausschöpft. Dafür lohnt es sich auch, kleine Nachteile in Kauf zu nehmen, die später beseitigt werden, wie z.B. beim bereits erwähnten ASL-Requester.

Probleme treten auch durch die höheren Prozessoren auf. Beim MC68000 war selbstmodifizierender Code möglich, da die CPU keinen Befehlscache kannte. Bei Prozessoren mit Cache kann dies sehr leicht zu Abstürzen führen, da der Befehl zwar im Speicher abgeändert wurde, im Befehlscache des Prozessors jedoch nicht. Ebenso verbietet es sich, das erste Byte von Zeigern zu benutzen, auch wenn dies beim MC68000 möglich war. Inzwischen ist der Adreßbereich über die 24-Bitgrenze gewachsen und es gibt Speichererweiterungen für Turbokarten, die sich ab der Adressse $08000000 einblenden.

Des weiteren sind zwar einige Parameter von Funktionen in der Dokumentation enthalten, wurden jedoch nicht genutzt. Initialisieren Sie solche Parameter deshalb immer mit null. Für C-Programmierer ist das eine Selbstverständlichkeit, mancher Assembler-Programmierer maß dem jedoch keine Bedeutung bei. Andere Parameter wurden vom System nicht korrekt getestet: In der Dokumentation stand zwar LONG, intern wurde aber nur mit einem WORD gearbeitet. Dies mag sich aber bei der nächsten Betriebssystemversion ändern. Halten Sie sich also genau an die Richtlinien und Dokumentationen von Commodore, auch wenn sich Commodore selbst nicht immer daran hält.

Es ist leicht gesagt, man möge sich an die Commodore-Richtlinien für Programmierer halten. Alle nicht dokumentierten oder besonders gekennzeichneten Einträge sind tabu. Und außerdem ahne man bitte zukünftige Entwicklungen voraus. Hierzu ein Beispiel aus der »locale.library«: Jeder Locale-Catalog sollte einen korrekten 2.0-Versionstring enthalten. Dieser ist folgendermaßen aufgebaut:

```
$VER: <name> <version>.<revision> (<date>).
```

Mit der »locale.library« V38 wurden noch Kataloge geöffnet, bei denen keine Revision angegeben war. Ab V39 werden nur noch Kataloge mit korrektem Versionstring geöffnet. Achten Sie also auch bei solchen »Kleinigkeiten« auf Korrektheit.

Da sich aber noch nicht alles über das Betriebssystem regeln läßt, ist man als Programmierer einfach manchmal gezwungen, seine eigenen Wege zu finden. Dabei sollte man sich wirklich sehr behutsam vortasten und nicht allzuvieles als feste Tatsachen voraussetzen. Wollte man beispielsweise den Rahmen um Stringgadgets verändern, so hatte man es unter OS2.0 noch mit einem Border zu tun, der um das Gadget gezeichnet wurde. Ab OS3.0 handelt sich um ein Image. Versuche, den vermeintlichen Border zu ändern, mußten natürlich zum Absturz führen.

Besonders, da es gerade für das AA-Chipset unzählige Commodities und Patches gibt, ist Vorsicht geboten. Häufig werden Betriebssystemroutinen gepatcht, um aus Cyclegadgets kleine Menüs auszuklappen, das Menü direkt am Mauspfeil anzuzeigen oder die Grafikdarstellung zu verbessern. So sollte man sich unbedingt das äußerst nützliche KCommodity (Shareware von Kai Iske) zulegen. Es verbindet Screen- und Mausblanker, Screenpromoter, Uhr und viele weitere Funktionen in einem.

Gut ist nicht mehr gut genug

OS2.0/3.0 hat nicht nur grafische Veränderungen gebracht. Die Tatsache, daß der Umfang des ROMs von 256 KByte auf 512 KByte verdoppelt wurde, spricht für sich. Die neuen Möglichkeiten sollten natürlich genutzt werden. Dazu gehören u.a. Kommunikation über ARexx, Hotkeys über die »commodities.library«, mehrsprachige Programme dank »Locale« und Einrichten eines Programmes auf Festplatte durch den Installer. Da andere Artikel in diesem Heft auf die Programmiersprache des Installers und die Möglichkeiten von

```
CreateMsgPort equ -666
CreateCxObj   equ -30
CxBroker      equ -36
AttachCxObj   equ -84
SetFilter     equ -120
InitHotkey
 lea   NewBroker1,a5
 move.l 4.w,a6
 jsr   CreateMsgPort(a6)
 move.l d0,20(a5)
 beq.s .exit
 move.l CxBase,a6
 move.l a5,a0
 moveq #0,d0
 jsr   CxBroker(a6)
 lea   Broker1,a0
 move.l d0,(a0)
 beq   Cx_FreePort
 moveq #1,d0 * CxFilter
 sub.l a0,a0
 sub.l a1,a1
 jsr   CreateCxObj(a6)
 tst.l d0
 beq   Cx_FreeBroker
 move.l d0,a2
 move.l d0,a0
 lea   Hotkey_Text,a1
 jsr   SetFilter(a6)

 moveq #8,d0 * CxCustom
 lea   Hotkey_Routine,a0
 sub.l a1,a1
 jsr   CreateCxObj(a6)
 tst.l d0
 beq.s Cx_FreeBroker
 move.l d0,a3
 move.l Broker1,a0
 move.l a2,a1
 jsr   AttachCxObj(a6)
 move.l a2,a0
 move.l a3,a1
 jsr   AttachCxObj(a6)
 moveq #1,d0
 bsr.s Cx_Activate
.exit
 rts
NewBroker1
 dc.b  5,0   * Version
 dc.l  BrokerName
 dc.l  BrokerTitle
 dc.l  BrokerText
 dc.w  0
 dc.w  4   * Show/Hide
 dc.b  0,0
 dc.l  0   * MsgPort
 dc.w  0 * Strukturende Broker1
 dc.l  0
Hotkey_Text
 dc.b  'ctrl q',0
BrokerName
 dc.b  '1200-Demo',0
 even
*-------
ActivateCxObj equ -42
Cx_Activate
 move.l CxBase,a6
 move.l Broker1,a0
 jsr   ActivateCxObj(a6)
 rts
*-------
DeleteMsgPort  equ -672
RemoveCxObj  equ -102
DeleteCxObjAll equ -54
 FreeHotkey
 move.l CxBase,a6
Cx_FreeBroker
 move.l Broker1,a0
 jsr   RemoveCxObj(a6)
 move.l Broker1,a0
 jsr   DeleteCxObjAll(a6)
Cx_FreePort
 move.l 4.w,a6
 lea   NewBroker1,a5
 move.l 20(a5),a0
 jsr   DeleteMsgPort(a6)
 rts
*-------
GetMsg equ -372
ReplyMsg equ -378
CxMsgID equ -150
 GetCxMsg
 lea   NewBroker1,a0
 move.l 20(a0),d0
 beq.s .exit
 move.l d0,a0
 move.l 4.w,a6
 jsr   GetMsg(a6)
 move.l d0,d6
 beq.s .exit

 move.l d0,a0
 move.l CxBase,a6
 jsr   CxMsgID(a6)
 move.l d0,d2
 move.l 4.w,a6
 move.l d6,a1
 jsr   ReplyMsg(a6)
 cmp.l #15,d2
 beq.s .disable
 cmp.l #17,d2
 beq.s .enable
 cmp.l #19,d2
 beq.s .appear
*  cmp.l #21,d2
*  beq.s .disappear
 cmp.l #23,d2
 bne.s .exit

.kill
 moveq #-1,d7
 bra.s .exit

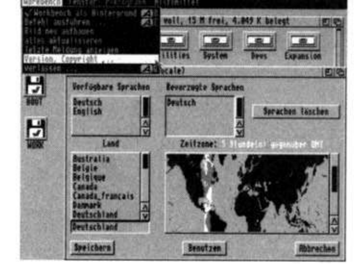
.disable
 moveq #0,d0
 bsr   Cx_Activate
 bra.s .exit

.enable
 moveq #1,d0
 bsr   Cx_Activate

.exit
 rts

.appear
 bsr   ShowAmigaGuide
 bra.s .exit
```


Listing 2: Programmierung von Commodities



ARexx eingehen, widmen wir uns vor allem den anderen Themen und da zuerst den sog. Hotkeys.

Sicher kennen Sie das Programm »FKey« aus der Commodities-Schublade. Damit können Sie bestimmten Tastenkombinationen Funktionen zuweisen, z.B. können Sie Programme oder ARexx-Scripts per Tastendruck starten, Fenster und Bildschirme manipulieren oder ganze Zeichenketten in Texte einfügen lassen. Ähnliches verwenden auch andere Programme, die per Tastendruck »sichtbar« werden, also z.B. ein Fenster öffnen, in dem Voreinstellungen getroffen werden können. Alle diese Programme nennt man Commodities.

System erweitern mit den Commodities

Die Programmierung eigener Commodities ist recht einfach. Wichtigste Voraussetzung ist die »commodities.library«. In ihr finden sich die wichtigen Funktionen zum Anlegen, Verwalten und Entfernen von Commodities. Ein Commodity, wie wir es in unserem Beispiel benötigen, besteht aus einem Broker und mehreren untergeordneten CxObjects. Ein Broker ist die wichtigste Struktur eines jeden Commodities. Ohne diese Struktur ist ein Programm kein angemeldetes Commodity und wird auch nicht vom Cx-Exchange angezeigt. Sie wird über die NewBroker-Struktur initialisiert, in der wichtige Informationen wie Name und kurze Beschreibung enthalten sind. Der wichtigste Eintrag ist jedoch der Messageport, den wir vorher mit »CreateMsgPort()« aus der »exec.library« angelegt haben.

Über diesen bekommt unser Programm Nachrichten geschickt, die von Exchange gesendet werden. Wenn Sie Exchange starten, werden im Fenster alle laufenden Commodities nebst Informationen gezeigt. Außerdem können Sie über die Gadgets Commodities entfernen, aktivieren oder deaktivieren, wenn das Commodity dies vorsieht. Unser Demoprogramm soll das natürlich alles beherrschen.

Nachdem wir den Broker jetzt per »CxBroker()« angelegt haben, hängen wir ihm noch zwei CxObjects an. Diese Objekte werden über »CreateCxObject()« initialisiert und erledigen die eigentliche Arbeit. Das erste ist vom Typ CxFilter und überwacht die in »SetFilter()« angegebene Tastenkombination. Das letzte Glied in der Kette ist ein CxObject vom Typ CxCustom. Ihm wird ein Zeiger auf die Routine übergeben, die angesprungen werden soll, sobald der Hotkey gedrückt wird. Über »AttchCxObj()« werden unsere drei Objekte miteinander in Verbindung gebracht und mit »ActivateCxObj()« aktiviert.

Zur Veranschaulichung dient Listing 2. Es ist nur ein Ausschnitt aus dem gesamten Programm, das sich auf der PD-Diskette zu dieser Ausgabe von »Faszination Programmieren« befindet (siehe S. 114).

Freizugeben ist der Broker ganz einfach. Zuerst wird er aus der Liste der Commodities entfernt und dann der gesamte Speicher wieder freigegeben. Natürlich müssen wir auch den Messageport wieder freigeben. Aber kümmern wir uns zuerst um die Kommunikation der Objekte. Wird der Hotkey gedrückt, wird nicht etwa eine Message verschickt, sondern die angegebene Routine direkt angesprungen. Wichtig ist, daß die Routine alle verwendeten Register rettet. Die Werte der Register beim Routinenanfang sind nicht dokumentiert.

Messages werden nur, wenn das Commodity über Exchange gesteuert wird. Um eine Nachricht auswerten zu können, holen wir sie uns einfach mit »GetMsg()« der »exec.library«. Dann finden wir mit »CxMsgID()« ihren Typ heraus und schicken sie wieder zurück. So einfach geht das. Etwas kompliziert wird es, will man mehrere Hotkeys verwenden. Aber auch das ist realisierbar. Sie müssen dem Broker lediglich neue CxObjects hinzufügen.

Was sich über die »commodities.library« noch gut realisieren läßt, sind Inputevents aller Arten, wie Mausbewegungen oder ein Tastendruck. Ob man nun Events verschlucken, neue hinzufügen oder aufgetretene Events konvertieren will, alles ist möglich.

Interessant ist auch die Fernsteuerung via ARexx, auf die wir im folgenden ganz kurz eingehen möchten. Auch hier erfolgt die Kommunikation über Messages und Ports. Sie müssen dem Port hier jedoch einen Namen geben und ihn über »AddPort()« in die Liste der öffentlichen Ports einbinden. Wie sie ARexx-Messages auslesen, sehen Sie im entsprechenden Listing 3. Unser 1200erDemo erkennt nur die Befehle »SCREENTOFRONT« und »ENDE«. Ein ARexx-Script, das unser Demo von außen steuert, sollte mit »ADDRESS 1200DEMO_REXX« beginnen. Ansonsten sollten nur die ARexx-Direktiven und die beiden eigenen Befehle verwendet werden.

Programmierer werden sich mehr und mehr fragen, wie viele Sprachen sie noch lernen müssen, um den Amiga komplett auszureizen? Neben der favorisierten Sprache wie Assembler oder C verwenden AmigaGuide, ARexx und Installer eine eigene Sprache. Da an anderer Stelle in diesem Heft genauer auf ARexx und den Installer eingegangen wird, hier nur der Hinweis, daß zu jedem größeren Programmpaket einfach ein Installer-Script dazugehört. Ansonsten steht der Benutzer entweder im Regen, weil er nicht weiß, welche Dateien wohin gehören, oder er muß alles per Hand kopieren.

MultiView - einer für alles.

Ebenso wichtig ist auch eine gute Dokumentation. Dazu bietet sich das AmigaGuide-Format von Commodore geradezu an. In der Regel wird ein solcher Guide über sein Default-Tool Multiview aufgerufen. Es ist jedoch auch möglich, AmigaGuides aus dem laufenden Programm zu laden und anzuzeigen, ja sogar von außen zu steuern.

Ein jeder Guide besteht aus einer ASCII-Datei, die mit der Kennung »@DATABASE <name>« beginnt. Wobei für <name> der Name des Guides zu setzen ist. Eingeteilt ist ein jeder Guide in Kapitel – sog. Nodes. Jede Node hat einen Namen, einen Titel und einen Text, der angezeigt werden soll. Die Node, die als erstes dargestellt werden soll, erhält immer den Namen MAIN.

```
AddPort   equ -354
CreateMsgPort equ -666
LN_NAME   equ 10

InitARe
xx move.l  4.w,a6
 jsr   CreateMsgPort(a6)
 lea   ARexxPort,a0
 move.l  d0,(a0)
 beq.s .exit
 move.l  d0,a1
 lea   ARexxPortName,a0
 move.l  a0,LN_NAME(a1)
 jsr   AddPort(a6)
.exit
 rts
ARexxPort
 dc.l  0
ARexxPortName
 dc.b  '1200DEMO_REXX',0
Blabla_Text
 dc.b  'SCREENTOFRONT',0
Ende_Text
 dc.b  'ENDE',0

 even
RemPort   equ -360
DeleteMsgPort equ -672

FreeARe
xx move.l  ARexxPort,d2
 beq.s .exit
 move.l  4.w,a6
 move.l  d2,a1
 jsr   RemPort(a6)
 move.l  d2,a0
 jsr   DeleteMsgPort(a6)
.exit
 rts

GetMsg equ -372
ReplyMsg equ -378
rm_ARG0  equ 40

GetARexxMsg
 move.l  ARexxPort,d0
 beq.s .exit

 moveq #0,d2

 move.l  4.w,a6
 move.l  d0,a0
 jsr   GetMsg(a6)
 tst.l d0
 beq.s .exit

 move.l  d0,a2
 move.l  rm_ARG0(a2),a3

 lea   Blabla_Text,a0
 bsr.s CmpString
 tst.l d0
 beq.s .next
 bsr   Hotkey_Routine
 bra.s .reply

.next
 lea   Ende_Text,a0
 bsr.s CmpString
 tst.l d0
 beq.s .reply
 moveq #-1,d7

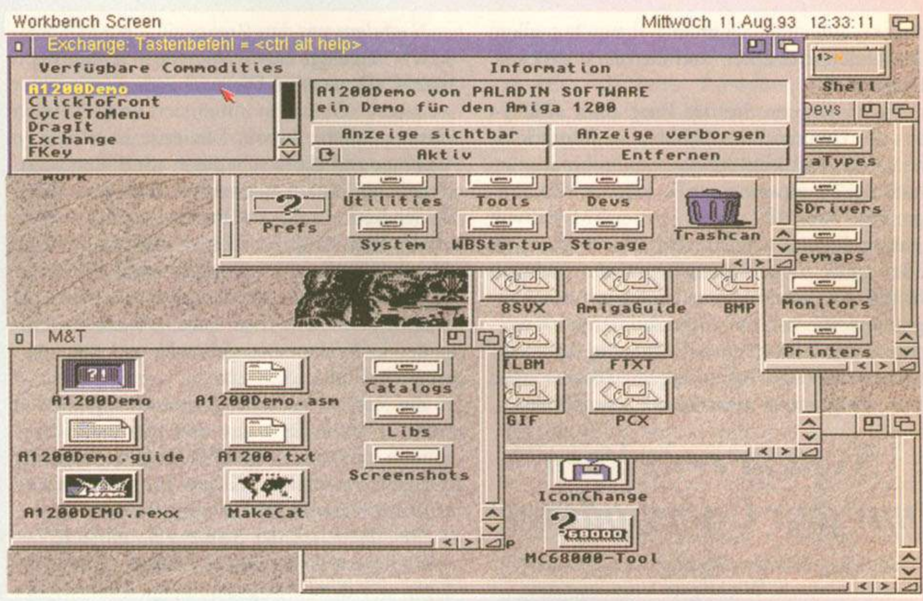
.reply
 move.l  a2,a1
 jsr   ReplyMsg(a6)
.exit
 rts

CmpString
 move.l  a3,a1
.loop
 move.b  (a0)+,d0
 beq.s .null
 bclr  #5,d0
 cmp.b (a1)+,d0
 beq.s .loop
.bad
 moveq #0,d0
 rts
.null
 tst.b (a1)
 bne.s .bad
 moveq #-1,d0
 rts
```


Listing 3: ARexx-Port selbstgestrickt

OS2.1 & 3.0





»CX-Exchange«: Hier können alle dem System angemeldeten Commodities angezeigt, an- und abgeschaltet oder entfernt werden

Initialisiert wird die Node folgendermaßen:
`@NODE <name> "<titel>"`
Alle folgenden Zeilen bis »@ENDNODE« sind darzustellender Text.

Besonderheiten beginnen immer mit einem Klammeraffen (»@«). Äußerst nützlich sind dabei die Links, also Querverweise, die durch Gadgets dargestellt werden, und zu anderen Kapitel verzweigen. Ein solcher Link könnte folgendermaßen aussehen:
`@{" Knopftext " LINK Kapitel3}.`
Als Kapitel kann auch eine Node eines anderen Guides angegeben werden. Dann steht der Guidename vor dem Nodenamen und wird durch »/« getrennt.

Im Listing 4 finden Sie ein Beispiel zum Öffnen eines Guides. Dieser muß im gleichen Verzeichnis wie das Programm liegen. Das Öffnen ist einfach: Der »amigaguide.library« wird eine NewAmigaGuide-Struktur übergeben. Den Rest übernimmt die Library. Ein Unterschied zu Multiview ist für den Benutzer nicht mehr feststellbar. Das heißt auch, daß nicht unbedingt ein AmigaGuide geöffnet werden muß, denn bekanntlich beherrscht Multiview ja alle Datatypes, von Text über Sound bis hin zur Grafik.

```
ShowAmigaGuide
 move.l  AgBase,d0
 beq.s .exit
 move.l  DosBase,a6
 jsr  GetProgramDir(a6)
 lea  MyNag_Lock,a0
 move.l  d0,(a0)
 beq.s .exit
 move.l  AgBase,a6
 lea  MyNag,a0
 sub.l a1,a1
 jsr  OpenAmigaGuideA(a6)
 move.l  d0,a0
 jsr  CloseAmigaGuide(a6)
.exit
 rts
MyNag
MyNag_Lock  dc.l 0
MyNag_Name  dc.l NagName
MyNag_Screen  dc.l 0
MyNag_ScreenName dc.l PubName
MyNag_Port  dc.l 0
MyNag_ClientPort dc.l 0
MyNag_BaseName dc.l BaseName
MyNag_Flags  dc.l 0
MyNag_Context  dc.l 0
MyNag_Node  dc.l NodeName
MyNag_Line  dc.l 1
MyNag_Extens  dc.l 0
MyNag_Private  dc.l 0
NodeName
 dc.b 'MAIN',0
NagName
 dc.b 'txt:m&t/1200Demo.guide',0
BaseName
 dc.b 'Demo.guide',0     © 1993 M&T
```

Listing 4: Demonstration der neuen AmigaGuides

Benutzt man zum Öffnen die Funktion »OpenAmigaGuideA()«, wird mit der Programmausführung so lange gewartet, bis der Benutzer den Guide wieder schließt. »OpenAmigaGuideAsyncA()« dagegen startet ein eigenes Programm und ihr eigenes kann inzwischen weiter arbeiten. Mit dem geöffneten Guide kann man dann noch über die »amigaguide.library« kommunizieren.

In unserem Beispiel lassen wir den Benutzer sogar wählen, welches Kapitel der Dokumentation er betrachten möchte. Dafür sorgt der Eintrag »Node« der NewAmigaGuide-Struktur. Neben dem Kapitel kann übrigens auch noch die Zeile angegeben werden. Voreingestellt ist die Main-Node und die erste Zeile.

Locale – man spricht Deutsch

Und noch ein letztes Feature, das in Zukunft alle Programme unterstützen sollten: die »locale.library«. Mit ihr kann der Benutzer wählen, in welcher Sprache sein Computer laufen soll. Das Betriebssystem und die Originalprogramme von Commodore sind inzwischen fast alle localefähig. Ein einfaches Einstellen mit dem Prefs-Programm »Locale« genügt. Viele Anwenderprogramme unterstützen diese Möglichkeit jedoch nicht. Vielleicht, weil die Programmierer denken, daß die Programme dann erst ab OS2.1 laufen. Doch weit gefehlt: Entsprechende Programme laufen sogar unter AmigaDOS1.2. Sie sind natürlich dann nicht localefähig, aber aufwärtskompatibel.

Grundvoraussetzung für »mehrsprachige« Programme bilden die Kataloge. Diese können Sie entweder mit MakeCat oder CatComp erstellen. Da der Katalog für unser Demoprogramm direkt mit dem Assembler-Listing und MakeCat erstellt wird, befindet sich eine Version von MakeCat ebenfalls auf der PD-Diskette zum Heft. Für jede Sprache muß ein eigener Katalog angelegt werden. Fehlt ein Katalog, so wird entweder ein anderer geöffnet oder einfach Englisch als Sprache verwendet. Wie man einen Katalog öffnet, schließt und sich die Stringadressen besorgt, zeigt Listing 5 (übernächste Seite). Besonders einfach geht dies mit der Unterroutine »GetString« und dem Macro »loclea«. Dieses ist voll kompatibel zu »lea«, jedoch natürlich nicht ganz so schnell, weshalb man es in Schleifen so sparsam wie möglich verwenden sollte. Zu beachten ist nur noch, daß diese Strings read-only sind.

Programme, die explizit auf die AA-Chips zugreifen, können auch einen MC68020 als Prozessor voraussetzen und ausreizen. Um andere Programme zusätzlich zu beschleunigen, können Sie natürlich bei zeitkritischen Routinen den Prozessortyp abtesten und entsprechend verzweigen. Besonders bei kurzen Schleifen (bis zu 256 Byte Code), die oft durchlaufen werden, ist dies sehr sinnvoll. Denn erstens lohnen sich verschiedene Versionen von Routinen nicht, wenn sie nur einmal durchlaufen werden und man nur ein paar Taktzyklen spart. Und zweitens wird die Ausführung dadurch wesentlich beschleunigt, daß die Befehle aus dem Befehlscache gelesen werden können, der beim MC68020 eben 256 Byte groß ist. Beim MC68040 ist er sogar 4 KByte groß, was eine weitere Beschleunigung mit sich bringt. Das verbietet natürlich selbstmodifizierenden Code, aber das dürfte im Vergleich zur Leistung ein nur geringer Nachteil sein.

MC68020 – das neue Herz des Amiga

Ebenfalls neu gegenüber dem MC68000 sind auch die Register VBR, SFC, DFC, A7" (oder MSP), sowie die Cache-Register CACR und CAAR. Diese dürfen jedoch nur im Supervisor-Modus benutzt werden. Apropos Supervisor: es sei hier noch einmal kurz an die einzige Aufwärts-Inkompatibilität erinnert. Das Statusregister darf vom MC68020 aufwärts nur noch im Supervisor-Modus gelesen werden (siehe oben). Interessant ist auch, daß die Adreßregister endlich auch mit ungeraden Adressen arbeiten können, so daß der Guru $80000003 fast der Vergangenheit angehört.

Denn Befehle müssen weiterhin an geraden Speicherstellen beginnen. Die von der »dos.library« verwendeten BCPL-Zeiger müssen natürlich weiterhin an durch vier teilbaren Adressen liegen, da dies Software-bedingt ist.

Besonders interessant und zeitsparend sind die neuen Adressierungsarten, die der MC68020 beherrscht. Wenig Neues bieten folgende zwei Möglichkeiten, die schon der MC68010 beherrschte. Mit ihnen wurde die Adreßdistanz der indirekten Adressierungsarten von 8, bzw. 16 Bit auf 32 Bit erweitert. Außerdem lassen sich jetzt Register skalieren, d.h. intern mit einem Faktor 1,2,4 oder 8 multiplizieren, ohne daß die Registerinhalte verändert werden. Ein Beispiel dafür wäre folgender Befehl:

```
move.w  #1000,($10000,a0,d2.w*8)
```

Er faßt sozusagen folgende 68000er-Befehle zusammen, ohne die Register zu verändern:

```
mulu    #8,d2
add.l   d2,a0
add.l   #$10000,a0
move.w  #1000,(a0)
```

Noch leistungsstärker, aber auch etwas komplizierter, sind die komplexen Adressierungsarten. Dazu ein ganz einfaches Beispiel: Statt

```
move.l  (a0),a0
clr.b   (a0)
```

können Sie jetzt

```
clr.b   ([a0])
```

schreiben, ohne daß a0 verändert wird. Dank vieler optionaler Parameter können diese Adressierungsarten eben sehr komplex werden. Alle Parameter in eckigen Klammern bilden zusammen einen Zeiger auf eine Speicherstelle, die dann wiederum indirekt adressiert werden kann. Dazu folgendes Beispiel:

```
tst.b   ([$10000,a0,d7.l*2],10)
```

Für den MC68000 ließe sich dieser Befehl so umschreiben:

```
add.l   d7,d7
add.l   d7,a0
add.l   #$10000,a0
tst.b   10(a0)
```

Alle vier komplexen Adressierungsarten finden Sie in der Tabelle 1, ebenso wie die zwei neuen indirekten Varianten. Doch nicht nur die Offsets wurden auf 32-Bit-Breite erweitert. Da jetzt auch der interne Datenbus 32 Bit breit ist, lassen sich auch Langworte leichter multiplizieren und dividieren. So multipliziert »muls.l d0,d1« beide Register mit voller 32-Bit-Breite. Dabei kann es leicht zu einem Overflow kommen. Deswegen kann auch »muls.l d0,d1:d2« benutzt werden. Dann befinden sich die oberen 32 Bit des 64 Bit großen Ergebnisses in d1, die unteren in d2. Ein Overflow ist somit nicht mehr möglich. Ebenso arbeiten die Befehle »mulu.l«, »divs.l« und »divu.l«.

Tabelle 1: die neuen Adressierungsarten

Mnemonik.	Art
(d1,An,Ri.s*scl)	ARI. [+Offset32]. [+Skalierung]
(label,pc,Ri.s*scl)	pc-relatic [+Offset32] [+Skalierung]
([d1,An],Ri.s*scl,d2)	doppelt indirekt, Post-Index
x([d1,pc],Ri.s*scl,d2)	doppelt indirekt, Post-index, pc-relativ
([d1,An,Ri.s*scl],d2)	doppelt indirekt, Pre-Index
x([d1,pc,Ri.s*scl],d2)	doppelt indirekt, Pre-Index, pc-relativ

.) Erklärungen zu Mnemonik:

d1/d2 = 32-Bit-Offset
An = beliebiges Adreßregister
Ri.s = beliebiges Register mit Größenangabe (word oder long)
scl = Skalierung (multipliziert den Registerinhalt intern mit 1,2,4 oder 8)
pc = Programm Counter

.) ARI = AdreßRegister Indirekt (entspricht z.B. (a0))

.) Angaben bei Art in eckigen Klammern sind optional, können also weggelassen werden. Bei den letzten vier Adressierungsarten unter Mnemonik sind praktisch alle Bestandteile optional '([])' ist also erlaubt !

Neue Befehle des 68020ers

Sicher nicht nur für Hochsprachen-Compiler interessant ist der Befehl »rtd«. Er arbeitet genau wie »rts«, hat jedoch einen Parameter. Mit »rtd #4« wird der Stack um vier erhöht und dann erst zurückgesprungen. Er entspricht also »addq.l #4,sp« und »rts«.

Ähnlich vereinigt auch der neue Befehl »extb.l dx« zwei Kommandos in sich. Er erweitert nämlich ein Datenregister vorzeichenrichtig von 8 auf 32 Bit. Bisher war dies nur über die getrennten Befehle »ext.w dx« und »ext.l dx« möglich.

Viel Arbeit ersparen auch die sogenannten Bitfield-Befehle. Man kann mit ihnen mehrere Bits auf einmal beeinflussen, ohne sich an Byte-, Word- oder Longwordgrenzen zu halten. So entsprechen »bfchg«, »bfclr«, »bfset« und »bftst« genau ihren Ein-Bit-Kollegen. Syntaktisch gibt es jedoch Unterschiede: So stehen die Bitnummern nicht vor der Adresse, sondern in geschweiften Klammern dahinter. Das folgende Beispiel setzt die Bits 3 bis 9 von d0. ungewöhnlich ist dabei die Numerierung der Bits. Hier wird von links nach rechts gezählt, also bei Bit #31 begonnen, das die Nummer 0 erhält. Das ist verständlich, wenn man bedenkt, daß man nicht an die üblichen Byte-Grenzen gebunden ist, wenn Bits im Speicher manipuliert werden.

Da wir von links zählen ist Bit #9 von d0 das 22. Bit des Feldes. Die Länge des zu setzenden Bitfeldes ist 7 Bits. Also heißt der Befehl:

```
bfset  d0{22:7}.
```

Mit der gleichen Syntax arbeiten »bfclr«, »bfchg« und »bftst«. Zum Kopieren ganzer Bitfelder stehen drei weitere Befehle zur Verfügung: Mit »bfexts« und »bfextu« werden Bitfelder aus dem Speicher in ein Datenregister geschrieben, wobei je nach Befehl das erste Bit als Vorzeichenbit beachtet wird. Umgekehrt

```
loclea
 macro
 pea   \1
 bsr   GetString
 move.l  (sp)+,\2
 endm

OpenLocale equ -156
OpenCatalogA equ -150

OpenCatalog
 move.l  LocBase,d0
 beq.s .exit
 move.l  d0,a6
 sub.l a0,a0
 jsr   OpenLocale(a6)
 lea   Locale1,a0
 move.l  d0,(a0)
 beq.s .exit

 move.l  d0,a0
 lea   CatName,a1
 sub.l a2,a2
 jsr   OpenCatalogA(a6)
 lea   Catalog1,a0
 move.l  d0,(a0)

.exit
 rts

Locale1
 dc.l  0
Catalog1
 dc.l  0
CatName
 dc.b  '1200Demo.catalog',0
 even

*-------

CloseCatalog equ -36
CloseLocale  equ -42

FreeCatalog
 move.l  LocBase,d0
 beq.s .exit
 move.l  d0,a6
 move.l  Catalog1,a0
 jsr   CloseCatalog(a6)
 move.l  Locale1,a0
 jsr   CloseLocale(a6)
.exit
 rts

*-------

GetCatalogStr equ -72

GetString
 movem.l d0-d1/a0-a1/a6,-(sp)
 move.l  LocBase,d0
 beq.s .Ende
 move.l  d0,a6
 move.l  Catalog1,d0
 beq.s .Ende
 move.l  24(sp),a1   * String
 moveq #0,d0
 move.l  a1,d1
 lea   LOCALE_START,a0
 sub.l a0,d1
 beq.s .GetIt
.Loop
 tst.b (a0)+
 bne.s .Ok
 addq.l  #1,d0
.Ok
 subq.l  #1,d1
 bne.s .Loop
.GetIt
 move.l  Catalog1,a0
 jsr   GetCatalogStr(a6)
 move.l  d0,24(sp)
.Ende
 movem.l (sp)+,d0-d1/a0-a1/a6
 rts     © 1993 M&T
```

Listing 5: Locale in voller Aktion



kann man auch aus dem Register ein Feld wieder in den Speicher scheiben. Hier gibt es nur einen Befehl, da das Vorzeichen des Registers im Bitfeld keine Rolle spielt. Der Befehl heißt »bfins«.

Ein letzter, etwas ausgefallenerer Befehl ist abgekürzt »bfffo« und steht für »bit field - find first one«, als finde die erste »1« eines Bitfelds. Die Nummer dieses Bits wird dann in ein Datenregister geschrieben. Ist das Bitfeld leer, so steht die Breite des Bitfeldes plus 1 im Datenregister.

Hier noch einige Tricks speziell für Assemblerprogrammierer. Besonders beliebt sind sog. Peep-Hole-Optimisations, also Guckloch-optimierungen, die nur einen Befehl verbessern und immer gültig sind. Am bekanntesten ist wohl die Verwendung der Quick-Befehle, z.B. moveq, addq und subq. Im direkten Gebrauch werden sie von einigen Assemblern automatisch verwendet. Schwieriger sind jedoch andere Fälle, bei denen der Sinn einer Quick-Anweisung nicht ganz so offensichtlich ist. Statt des – einfach aussehenden – Befehls

```
add.l   #100,d0
```

kann man auch

```
moveq   #100,d1
add.l   d1,d0
```

schreiben, was sowohl vom Code her kürzer als auch schneller von der Ausführung ist.

Noch schneller geht dies mit Adreßregistern. Hier kann man statt

```
add.l   #100,a0
```

auch nur Word-Breite verwenden, da Adreßregister automatisch auf Langwort-Breite erweitert werden:

```
add.w   #100,d0
```

Noch schneller ist jedoch

```
lea     100(a0),a0
```

Durch oben erwähnte Eigenschaft ist auch eine Art »moveq« für Adreßregister möglich. Nicht ganz so schnell, aber dafür mit voller Wort-Breite:

```
move.w #1000,a0.
```

Darf es noch etwas schneller sein?

Diese Optimierungen arbeiten natürlich auch mit der Subtraktion. Im Gegensatz zu diesen Befehlen sind die Multiplikations- und Divisionsbefehle äußerst langsam (bis zu 140 Taktzyklen) und deshalb bis zu 35mal langsamer als andere Befehle. Vor jeder Multiplikation sollte deshalb getestet werden, ob einer der Faktoren Null oder Eins ist, was eine Menge Rechenarbeit erspart.

Mit den Bitschiebebefehlen in Kombination mit Addition und Subtraktion kommt man oft schneller ans Ziel. Dies ist vor allem in Schleifen wichtig, die oft durchlaufen werden und/oder besonders zeitkritisch sind. Hier empfiehlt es sich, so viel Rechenarbeit wie möglich vor dem Schleifenrumpf zu erledigen und die Ergebnisse eventuell in Registern zu speichern. Das Registerretten am Anfang und Ende der Schleifen geht bestimmt schneller, als eine ständige Neuberechnung.

Allgemein bekannt ist inzwischen auch, daß der Befehl »clr« nur in Ausnahmefällen auf Register verwendet werden sollte, nämlich dann, wenn nur ein einziges Byte oder Word gelöscht werden soll. Für Datenregister benutzt man »moveq #0,dx« und da Adreßregister sowieso nicht zu den Adressierungsarten des »clr«-Befehls gehören, wurde folgende Lösung gefunden:

```
sub.l ax,ax
```

Rückkehrbefehl: kurz aber unnötig

So kurz der Befehl »rts« auch ist, oft kann er eingespart werden. Wenn direkt davor ein »bsr« oder »jsr« steht, kann man diese in »bra«, bzw. »jmp« umwandeln. Dadurch spart man 26 Buszyklen und außerdem zwei Byte Code. Auch wenn sich dies nicht viel anhört, so können diese beiden fehlenden Bytes weitere indirekte Verkürzungen nach sich ziehen. Denn je kompakter der Code ist, desto häufiger können Kurzformen für Sprungbefehle angewandt werden, was wieder zu einer Verkürzung führen kann, usw.

Noch ein Trick mit dem »rts«-Befehl: Wenn Sie eine Routine anspringen wollen, deren Adresse Sie erst im Verlauf des Programms bestimmen oder errechnen, können Sie dies mit »jmp (a0)« verwirklichen. Wenn Sie jedoch alle Register für die Übergabe von Daten benötigen, hilft nur ein eleganter Trick. Legen Sie die Sprungadresse mit »pea a0« auf den Stack, schieben Sie Ihren Übergabewert nach a0 und verwenden sie dann »rts«. Dieses holt die vermeintliche Rücksprungadresse vom Stapel und springt ihre Routine an. Wollen Sie »jsr (a0)« ersetzen, so müssen Sie vor der Adresse der

Tabelle 2: die neuen Befehle

Name	Operatoren	Funktion
BFCHG	<ea>{bo.:bfl.}	Bitfeld invertierten
BFCLR	<ea>{bo:bfl}	Bitfeld löschen
BFEXTS	<ea>{bo:bfl},dn	Bitfeld vorzeichenrichtig übertragen
BFEXTU	<ea>{bo:bfl},dn	Bitfeld vorzeichenlos übertragen
BFFFO	<ea>{bo:bfl},dn	Finde erste 1 im Bitfeld
BFINS	dn,<ea>{bo:bfl}	Bitfeld aus dn in den Speicher schreiben
BFSET	<ea>{bo:bfl}	Bitfeld setzen
BFTST	<ea>{bo:bfl}	Bitfeld testen
DIVS.L	<ea>,dn	vorzeichenbehaftete Division (32 Bit÷32 Bit)
DIVU.L	<ea>,dn	vorzeichenlose Division (32 Bit÷32 Bit)
MULS.L	<ea>,dn	vorzeichenbehaftete Multiplikation (32 Bit•32 Bit)
MULU.L	<ea>,dn	vorzeichenlose Multiplikation (32 Bit•32 Bit)
DIVS.L	<ea>,dr:dq	vorzeichenbehaftete Division (64 Bit÷32 Bit)
DIVSL.L	<ea>,dr:dq	vorzeichenbehaftete Division (32 Bit÷32 Bit)
DIVU.L	<ea>,dr:dq	vorzeichenlose Division (64 Bit÷32 Bit)
DIVUL.L	<ea>,dr:dq	vorzeichenlose Division (32 Bit÷32 Bit)
MULS.L	<ea>,dr:dq	vorzeichenbehaftete Multiplikation (32 Bit•32 Bit)
MULU.L	<ea>,dr:dq	vorzeichenlose Multiplikation (32 Bit•32 Bit)
PACK	-(Am),-(An),#x	bilde gepackte Dezimalzahl und addiere #
xPACK	Dm,Dn,#x	"
UNPK	-(Am),-(An),#x	bilde ungepackte Dezimalzahl und addiere #
xUNPK	Dm,Dn,#x	"
CHK2.x	grenzen,ea	wie chk, nur mit zwei Grenzen
CMP2.x	grenzen,ea	wie cmp, nur mit zwei Grenzen
EXTB.L	dn	erweitert dn.b direkt vorzeichenrichtig auf dn.l
RTD	#x	wie rts, addiert aber vorher #x auf den Stapel
MOVE	ccr,ea	schreibe Conditioncoderegister nach ea
BKPT	#x	Breakpoint Nummer #x setzen
CALLM	#x,ea	Aufruf eines Moduls mit Deskriptor
CAS.x	Dc,Du,ea	Wenn Dc=ea, dann Du->ea, sonst ea->Dc
CAS2.x	Dc1:Dc2,Du1:Du2,(Rn):(Rm)	wie CAS nur mit zwei Vergleichswerten
MOVEC.x	Rc,Rn	Lese Supervisor-Register
MOVEC.x	Rn,Rc	Lade Supervisor-Register
MOVES.x	Rn,ea	Lade alternativen Adreßbereich
MOVES.x	ea,Rn	Lese alternativen Adreßbereich
RTM	Rn	Rückkehr aus einem Modul und Löschen des Deskriptors
TRAPcc	ea	bedingter Sprung in eine Exception

.) erstes Bit des Feldes (0-31 oder Datenregister)
.) Größe des Bitfeldes in Bits (1-32)

Routine die wirkliche Rücksprungadresse auf dem Stack ablegen, zum Beispiel so:

```
 pea    WirklicheAdresse
 pea    SubRoutine
 rts
WirklicheAdresse
* ...
SubRoutine
* ...
 rts
```

Was weiter zu einer erhöhten Kompaktheit des Codes führt, ist die PC-relative Adressierung. Dadurch sparen Sie pro »move« oder »lea« zwei Byte ein, wenn Sie auf Daten innerhalb des Programms zugreifen. Außerdem werden Programme dadurch kürzer, daß im Reloc-Hunk, einer Tabelle mit 4-Byte-großen Offsets, ein Eintrag fehlt. Noch kürzer wird das Programm, wenn es total pc-relativ geschrieben wurde. Dann entfällt nämlich auch der Reloc32-Hunk.

Das Problem dabei ist, daß man keine PC-relativen Adressen als Ziel verwenden darf. Deshalb verwendet man besser

```
 lea      Ziel_Label(pc),A
x move.l  Rx,(Ax)
```

anstelle von

```
 move.l  Rx,(Label).l
```

Das ist genauso lang, vermeidet aber einen Eintrag im Reloc-Hunk. Ein weiterer Vorteil von 100% PC-relativen Programmen ist, daß sie beliebig im Speicher verschoben oder kopiert werden dürfen.

Wenn Sie Programme jetzt noch so schreiben, daß sie nie schreibend auf Daten zugreifen, die im Programm liegen, können Sie bei den Protection-Bits getrost das Pure-Bit setzen. Es bewirkt, daß Ihr Programm ständig im Speicher bleiben kann und nicht jedesmal nachgeladen werden muß. Nach diesem Prinzip arbeiten viele CLI-Befehle. Anstelle eines Datenblocks im Programm, muß dieser jedesmal neu im Speicher angelegt werden. Am besten definieren Sie Ihren Speicherbereich als eigene Struktur, auf die Sie über Offsets zugreifen.

Falls Sie in einer solchen oder anderen Struktur Flags verwenden wollen, also Boolean-Variablen, die entweder TRUE oder FALSE sind, so sollten Sie dafür keine einzelnen Bits verwenden, sondern Bytes. Zwar verbrauchen Bits natürlich weniger Speicher als Bytes. Dennoch ist die Methode mit den Bytes wesentlich kürzer, wenn die Flags untereinander keine bestimmte Beziehung haben. Sowohl das Setzen und Löschen als auch das Testen der Bytes ist schneller und kürzer als die Bitbefehle, da man keine Bitnummer angeben muß. Also lieber

```
 tst.b   Flag(Ax)
```

als

```
 btst    #0,Flag(Ax).
```

Generell ist beim Optimieren von Assemblercode immer wieder Phantasie gefragt. Denn nicht diese kleinen Optimierungen machen ein Programm so schnell, sondern die verwendeten Algorithmen. Denken Sie also schon vor dem Programmieren schwieriger Programmteile daran, eine möglichst effiziente Unterroutine zu schreiben.

Weil der Startupcode von Programmen immer wieder einen eventuellen Workbenchstart unberücksichtigt läßt, hier noch eine korrekte Variante. Wichtig ist vorallem das Zurückschicken der WBMsg. Denn dadurch wird der gesamte Programmspeicher wieder freigegeben. Wenn vorher nicht das Multitasking abgeschaltet wird, kann der Speicher überschrieben werden, in dem die letzen Befehle bis zum 'rts' stehen. Nachdem der Task beendet wurde, erlaubt das Betriebssystem das Multitasking natürlich wieder.

AA – viel Power mit zwei »A«

Auf den folgenden Seiten gehen wir nun auf die Programmierung der neuen Grafikfähigkeiten ein. Die erste Bekanntschaft mit den neuen AA-Chips macht man bereits beim Öffnen eines Screens, bzw. direkt davor. Schon bei der Wahl des Darstellungsmodus stoßen wir auf eine verwirrende Vielfalt von Möglichkeiten. Unter OS3.0 gibt es inzwischen (offiziell) neun verschiedene Darstellungstypen. Theoretisch kann sich jeder seinen privaten Anzeigemodus schreiben, doch sollten die Vorgegebenen in der Regel ausreichen. In unserem Beispiel öffnen wir einen HiRes-HAM8 Screen ohne besondere Klassifizierung des Modus (ID=$00008804). Auf einem normalen Amiga wird der Screen in PAL oder NTSC interlace geöffnet. Besitzer eines VGA-Monitors, die »Modus übernehmen« in den »IPrefs« angeschaltet haben, erhalten einen DblPAL bzw. DblNTSC Screen. Konnte man früher den HAM-Modus nur in den niedrigen Auflösungen nutzen, so gibt es diese Beschränkungen nun nicht mehr. Eine Ausnahme bildet der A2024 Modus: er kann natürlich keine Farben darstellen, da dies der Monochrom-Modus ist. Trotzdem stehen uns noch genug verschiedene Auflösungen zur Verfügung (siehe Tabelle Seite 16). Seitdem Commodore den neuen 1942 Multisync-Monitor ausliefert, existieren neue Monitortreiber, die extra auf diesen Monitor angepaßt wurden. Mit dem Mitsubishi EUM 1941A sehen diese Modi dann etwas seltsam aus: das Bild ist wesentlich zu klein, da sich die Frequenzen geändert haben. Außerdem sind zwei neue Auflösungen dazugekommen: von nun an kann man auch im Euro72- und Multiscan-Modus niedrige Auflösungen wie 320x200 nutzen. Welcher Modus am Ende überhaupt benutzt werden kann, hängt vom verwendeten Monitor ab. Genaue Angaben darüber finden Sie am Ende des Artikels.

Ab OS2.0 ist es möglich, Screens gleich beim Öffnen mit den richtigen Farben zu belegen. Dazu muß man dem System mit dem Tag »SA_Colors« eine ColorSpec-Liste übergeben. Da mit AA und OS3.0 die Farbwerte nicht mehr 12 Bit, sondern 24 Bit breit sind, sollte man diese Möglichkeit nicht mehr benutzen. Dafür gibt es jetzt das Tag SA_Colors32, mit dem ein ColorTable angegeben werden kann. Der Aufbau einer solchen Tabelle sieht dabei wie folgt aus:

Das erste Word der Tabelle gibt dem System an, wieviel Farben geändert werden sollen und das zweite Word gibt an, ab welcher Farbnummer dies geschehen soll. Nun folgen jeweils 3 Longwords pro Farbe mit den 32 Bit großen Rot-, Grün- und Blauwerten (RGB). Dabei ist zu beachten, das die einzelnen Farbwerte linksbündig sind. Obwohl der Amiga im Moment nur 8 Bit Farbwerte kennt, müssen alle 32 Bit angegeben werden. Statt $0000008f muß man also $8f8f8f8f angeben.

```
 1 WORD : Anzahl der Farben im Table (n)
 1 WORD : erste Farbnummer des Table
 n x 3 LONG : linksbündige Rot, Grün und
              Blauwerte der Farbe
```

```
Forbid      equ -132
FindTask    equ -294
WaitPort    equ -384
GetMsg      equ -372
ReplyMsg    equ -378
pr_MsgPort  equ 92
pr_CLI      equ 172
Start
   movem.l d0/a0,-(sp)
   moveq   #0,d7              * WBMsg
   move.l  4.w,a6
   sub.l   a1,a1
   jsr     FindTask(a6)
   move.l  d0,a0
   tst.l   pr_CLI(a0)
   bne.s   .VonCLI
.VonWB
   lea     pr_MsgPort(a0),a0
   jsr     WaitPort(a6)
   jsr     GetMsg(a6)
   move.l  d0,d7
.VonCLI
   movem.l (sp)+,d0/a0

StartProgramm
*** hier steht jetzt das ***
*** eigentliche Programm ***
Ende
   tst.l   d7
   beq.s   .exit
   move.l  4.w,a6
   jsr     Forbid(a6)
   move.l  d7,a1
   jsr     ReplyMsg(a6)
.exit
   moveq   #0,d0
   rts          ;     © 1993 M&T
```

Listing 6: Programmstart von der Workbench

Um zu verhindern, daß beim Bildaufbau (z.B. beim Verschieben eines Fensters) mit mehreren Bitplanes das bekannte Flackern auftritt, geben wir den Tag »SA_Interleaved« mit TRUE an. Bisher wurde jede Bitplane einzeln im Chip-Ram untergebracht. Dies konnte dazu führen, daß die einzelnen Bitplanes weit voneinander entfernt waren. Die neuen Interleaved-Bitmaps gehen deshalb einen anderen Weg. Alle Bitplanes werden als ein großer Block im Chip-Ram belegt. Die Besonderheit daran ist der Aufbau dieser Bitplanes: Alle Bitplanes werden wie ein Kartenspiel »gemischt«, d.h. erst die 1.Scanline der 1.Bitplane, dahin-



ter liegt dann gleich die 1.Scanline der 2.Bitplane, dann die der Dritten usw. Erst wenn alle Bitplanes ihre 1.Scanline angeordet haben, geht es mit der zweiten Scanline weiter. Wer dachte, daß die Bitmap-Struktur und die Bitplanes immer gleich bleiben (müssen), der wird sein blaues Wunder erleben. Dies bedeutet auch, daß man auf keinen Fall mehr die in der Screen-Struktur bei sc_BitMap (Offset 184) eingebettete Bitmap benutzt, da in der Zukunft deren Struktur in der Zukunft noch größer wird. Die richtige BitMap bekommt man besser über den RastPort des Screens. Außerdem wird sich der Aufbau der BitPlanes ebenfalls ändern. Man denke nur an die Chunky Pixel Modi von Grafikkarten.

Wenn wider Erwarten unsere Screens nicht geöffnet werden konnten oder der Rechner sogar abstürzt, so kann dies mehrere Ursachen haben: z.B. war die DisplayID falsch, weil der Monitortreiber nicht gesetzt ist oder das AA-Chip Set nicht aktiviert und deshalb der gewünschte HiRes-HAM8 Modus nicht möglich war. Beim Booten schaltet der Amiga in den ECS-Modus, um für Spiele, die über den Bootblock starten, kompatibel zu bleiben. Erst mit Hilfe des Befehls »SetPatch« werden die neuen Fähigkeiten aktiviert. Das bedeutet aber nicht, daß alle Programme, die über den Bootblock starten, AA nicht benutzen können. Für diese Fälle gibt es die Funktion »SetChipRev()« (Offset -888) der »graphics.library«, die mit dem Wert $1f in D0 aufgerufen wird und AA anschaltet. Da die Funktion nur einmal aufgerufen werden kann, sollte man sie nur für den Bootblock verwenden (in der Regel steht ja »SetPatch« in der »Startup-Sequence«).

Um die Farbverläufe dann in das Fenster zu bekommen, verwenden wir die Funktion »WritePixelArray8()«, da dies wesentlich schneller ist als zig-mal »SetAPen()« und »WritePixel()« aufzurufen... Diese Funktion schreibt die in einer Tabelle byteweise angegebenen Farbnummern als Pixel in die angegebene Fläche.

Soweit sollte alles klar sein. Doch wie bringt man nun den HAM8 Modus dazu, daß er genau das darstellt, was wir haben wollen?

Wie funktioniert HAM8?

Durch einfaches Setzen eines Punktes in einer mit »SetAPen()« gewählten Farbe geht es nicht. Wer sich noch nie mit dem HAM Modus befaßt hat, wird am Anfang Probleme haben seine Funktionsweise zu verstehen. Das Prinzip ist aber nicht so schwer wie es sich anhört. Die Schwierigkeit liegt darin, dieses Wissen so umzusetzen, daß am Ende auch das gewünschte Ergebnis herauskommt.

Die Funktionsweise der beiden HAM-Modi (HAM6 und HAM8) ist nahezu identisch. Der einzige Unterschied liegt in der Anzahl der verwendeten Bitplanes. Der HAM8-Modus hat zwar acht Bitplanes, jedoch besitzt seine Palette nur 64 statt 256 verschiedene Farben. Diese Farben kann man dafür wie gewohnt ansprechen, d.h. »SetAPen()« setzt mit den Werten 0 bis 63 (binär %00000000 - %00111111) die jeweilige Farbe der Palette und WritePixel() zeichnet einen Punkt in dieser Farbe. Die Besonderheit liegt in den beiden oberen Bits, den sog. Modify-Bits. Zusammen ergeben sie einen Modify-Wert zwischen 0 und 3. Ist dieser Null, dann setzt das System die durch die unteren sechs Bits angegebenen Farbe aus der Palette, also 0 bis 63. Bei einem anderen Modify-Wert (1 bis 3) kommt die Besonderheit des HAM-Modus zur Geltung. Dann wird die Farbe des linken Nachbarpixels übernommen, verändert und ergibt somit die Farbe des Punktes, der gesetzt wird.

Diese Veränderung ist der Knackpunkt des Ganzen, denn sie unterliegt einer strengen Regel: Es kann jeweils nur der Rot- oder Grün- oder Blauwert der Farbe des linken Pixels geändert werden, niemals aber zwei oder alle drei Werte gleichzeitig. Das hat zur Folge, das ein weißer Pixel nicht in einem Schritt zu einem roten Pixel modifiziert werden kann, da dazu der Grün- und Blauwert gleichzeitig auf Null gesetzt werden müßte. In einem solchen Fall müßte man eine Originalfarbe aus der Palette nehmen. Beim Modify-Wert 1 wird der ursprüngliche Blauwert durch den mit Hilfe der

Hires-HAM8 Bildschirm
Farbverlauf
Hilfe
AmigaGuide
Commodities
Locale
ARexx
PALADIN
Weich

»A1200Demo«: Unser Demoprogramm, das dieses Bild erzeugt hat, nutzt die speziellen Eigenschaften des neuen Betriebssystems und der Grafikchips. Sie finden es auf der PD-Diskette zum Heft mit kompletten Assembler-Quellcode.

unteren 6 Bits angegebenen Blauwert ersetzt. Gleiches gilt für einen Wert von 2 (Grün) und 3 (Rot).

Wenn soweit alles klar ist, sollte man eigentlich auch das Problem sehen, das dabei auftritt: Seit AA kann eine Farbe jeweils 256 verschiedene RGB-Abstufungen besitzen, also eine Farbtiefe von 3 x 8 Bit = 24 Bit. Doch im HAM8-Modus werden die einzelnen Farbwerte nicht durch 8-Bit- sondern nur durch 6-Bit-Werte ersetzt. Die unteren beiden Bits des jeweiligen Farbwerts bleiben dabei unangetastet. Man kann also nicht mehr alle Farbwerte setzen, sondern ist auf eine Modify-Tiefe von 6 Bit beschränkt.

An dieser Stelle sollte man gleich das Gerücht über Bord werfen, es können somit nur 262 144 Farben angezeigt werden. Das stimmt nicht. Die 64 Grundfarben in der Palette sind immer noch reine 24-Bit-Farben, bei denen auch die unteren zwei Bit gesetzt sein können. Bei kluger Wahl seiner Grundpalette hat man Farben mit allen 64 möglichen Kombinationen der untern 2 Bits in allen drei Grundfarben (2 hoch 2*3 = 64). Da diese beim Modifizieren erhalten bleiben und nicht gelöscht werden, können selbstverständlich wesentlich mehr Farben dargestellt werden (bei einer Auflösung von 1280 512 Pixeln sind dies bis zu 655 360 verschiedene Farben).

Genau diesen Trick verwenden wir nun in unserem Demoprogramm. Dabei machen wir uns eine Schwäche des Auges zunutze, das kleine Variationen einer Farbe nicht unterscheiden kann. Dem Auge ist es egal, ob die Farbe Rot nun aus $ff Rot, $00 Grün und $00 Blau zusammengesetzt ist, oder ob sie aus $ff Rot, $03 Grün und $03 Blau besteht. Da die unteren zwei Bit im HAM8 nicht modifiziert werden, benutzen wir einfach vier Grundfarben für unseren Farbverlauf: $00, $00, $00; $01, $01, $01; $02, $02, $02 und $03, $03, $03. Somit sind die unteren zwei Bit bei den Rot-, Grün- und Blauwerten immer gleich gesetzt und man kann durch geschicktes Modifizieren alle 265 Grautöne im HAM8 darstellen, wie im Bild zu sehen ist. Mit Hilfe des »Weich«-Gadgets kann man den Trick an- und abschalten. Wird er abgeschaltet, hat man nur noch 64 Grautöne und es treten leichte Farbsprünge auf, die man nun wieder sehen kann.

Natürlich gilt für den HAM6-Modus ähnliches wie für HAM8, die hier möglichen 16 Grundfarben sind ebenfalls 24-Bit Farben. Es können also weit mehr als 4096 Farben angezeigt werden.

Seit OS3.0 kann man mit Hilfe der Preferences den Standard »Busy Pointer« selber machen. Doch wird dieser nicht automatisch in eigenen Programmen vom System benutzt. Zum Setzen des BusyPointers oder eines anderen, beliebigen Mauspfeils gibt es drei neue Window-Tags:

Dem Tag »WA_Pointer« muß eine ExtSprite Struktur oder NULL angegeben werden. Bei NULL wird der System Mauspfeil gesetzt. Der Wartezeiger wird mit dem Bool-Tag »WA_BusyPointer« (TRUE) gesetzt. Mit »WA_PointerDelay« (ebenfalls ein Bool-Tag) teilen wir dem System mit, daß ein neuer Mauspfeil erst nach einer kurzen Verzögerung dargestellt wird, falls in der Zwischenzeit nicht eine erneute Änderung stattfindet. Dies wird häufig beim Setzen des BusyPointers benutzt, damit der Mauspfeil nicht erst die Busy-Grafik zeigt um sofort darauf wieder ein anderes Aussehen anzunehmen, falls die Wartezeit sehr kurz war. Diese Tags können bei »OpenWindowTagList()« und bei »SetWindowPointerA()« angegeben werden.

Jetzt bleibt noch die Frage zu beantworten, wie man sich eine ExtSprite-Struktur beschafft, damit man nicht nur die System-Pointer setzen kann? Am einfachsten geht dies mit Hilfe von Objekten. Dazu verwenden wir die Funktion »NewObjectA()«; ihr übergeben wir den Zeiger auf die Klasse des Objekts und eine Tagliste. Die Klasse wird mit Hilfe eines simplen String definiert, in unserem Fall »pointerclass«. Über die Tags geben wir alle wichtigen Daten an. Die BitMap, die das Aussehen des Pointers bestimmt, wird mit dem Tag »POINTERA_BitMap« gesetzt. Den Offset des Hotpoints geben wir mit »POINTERA_XOffset« und »POINTERA_YOffset« an.

Bisher waren Sprites 16 Pixel breit, also ein Word breit. Mit AA können es auch 32 (zwei Words) und 64 Pixel (vier Words) sein. Diese Breite geben wir mit Hilfe des Tags »POIN-

Tabelle 3: Bildschirmmodi

AA-Modus	VGAOnly	Auflösungen	Mode-ID	max.sichtbar	VGAOnly
NTSC		LoRes	=$00000000	362 x241	-
60 Hz	-	HiRes	=$00008000	724 x241	-
15,72 kHz	-	SuperHiRes	=$00008020	1448 x241	-
ID=$00011000		Interlace	=$00000004	1448 x482	-
PAL		LoRes	=$00000000	362 x283	-
50 Hz	-	HiRes	=$00008000	724 x283	-
15,60 kHz	-	SuperHiRes	=$00008020	1448 x283	-
ID=$00021000		Interlace	=$00000004	1448 x566	-
MULTISCAN		ExtraLoRes	=$00000000	164 x240	164 x248
58 Hz	60 Hz	LoRes	=$00008000	328 x240	328 x248
29,29 kHz	31,44 kHz	Productivity	=$00008020	656 x240	656 x248
		NonInterlace	=$00000004	656 x480	656 x495
ID=$00031000		Interlace	=$00000005	656 x960	656 x990
EURO72		ExtraLoRes	=$00000000	164 x200	164 x207
69 Hz	70 Hz	LoRes	=$00008000	328 x200	328 x207
29,32 kHz	31,43 kHz	Productivity	=$00008020	656 x200	656 x207
		NonInterlace	=$00000004	656 x400	656 x414
ID=$00061000		Interlace	=$00000005	656 x800	656 x828
EURO36		LoRes	=$00000000	362 x200	-
73 Hz	-	HiRes	=$00008000	724 x200	-
15,76 kHz	-	SuperHiRes	=$00008020	1448 x200	-
ID=$00071000		Interlace	=$00000004	1448 x400	-
SUPER72		LoRes	=$00000000	228 x307	228 x315
71 Hz	72 Hz	HiRes	=$00008000	456 x307	456 x315
23,21 kHz	24,62 kHz	SuperHiRes	=$00008020	912 x307	912 x315
ID=$00081000		Interlace	=$00000004	912 x614	912 x630
DbINTSC		LoRes	=$00000000	360 x226	360 x233
58 Hz	59 Hz	HiRes	=$00008000	720 x226	720 x233
27,66 kHz	29,02 kHz	NonInterlace	=$00000004	720 x452	720 x467
ID=$00091000		Interlace	=$00000005	720 x904	720 x934
DbIPAL		LoRes	=$00000000	360 x275	360 x282
48 Hz	50 Hz	HiRes	=$00008000	720 x275	720 x282
27,50 kHz	29,45 kHz	NonInterlace	=$00000004	720 x550	720 x564
ID=$000A1000		Interlace	=$00000005	720 x 1100	720 x 1128

Zusätzliche Farbanzeigetypen in allen Auflösungen:
ExtraHalfBrite = $00000080 (nur mit 6 Bitplanes)
HAM = $00000800 (nur mit 6 o. 8 Bitplanes)
DualPlayField = $00000400
DualPlayField2 = $00000440



TERA_WordWidth« an, und zwar, wie der Name sagt, nicht in Pixel sondern in Words (1, 2 oder 4). Ist die Breite der BitMap kleiner als die Wordbreite, wird der Rest mit Null-Bytes ausgefüllt, ist sie größer, wird der Rest der BitMap abgeschnitten. Für die Höhe bestehen keine Beschränkungen, diese wird einfach durch die BitMap angegeben. Nun gibt die Breite aber nicht automatisch auch die Auflösung an. Dies geschieht mit den beiden Parametern »POINTERA_XResolution« und »POINTERA_YResolution«.

Ein X-Wert von Null entspricht der ECS-Auflösung (LoRes, nur auf SuperHires-Screens HiRes). Eine »1« ist immer eine LoRes-Auflösung des Sprites, »2« immer HiRes und »3« SuperHiRes. Bei »4« paßt sich der Sprite der Auflösung des Screens an, auf dem er sich gerade befindet. Durch den Y-Wert gibt man die Anzahl der sichtbaren Pointerlines an. Diese macht sich allerdings erst auf 30-kHz-Screens bemerkbar, da auf den normalen Screens die Y-Auflösung immer LoRes ist (Y-Wert = »0«). Bei »2« erhöht sich die darstellbare Höhe des Sprites bei 30-kHz-Screens auf 400 bis 480 Lines. Setzt man den Wert »4« ein, paßt sich die vertikale Auflösung der des Screens an, wenn dies möglich ist.

Super-Sprites für Spiele

Am besten experimenieren Sie etwas mit den Werten und dem Aussehen des Pointers, da die Aufzählung oben sicherlich etwas trocken war (das Listing finden Sie auf der PD-Diskette zum Heft, siehe Seite 114).

Nun wollen wir noch etwas auf die neuen Hardwareregister eingehen, damit auch Spiele- und Demoprogrammierer ihrer Fantasie freien Lauf lassen können. Es sei hier jedoch gleich gesagt, daß diese Belegung von Commodore nicht offiziell ist. Es kann sein, daß sich die Belegung beim nächsten Chip-Set ändert. Wenn Sie sichergehen wollen, das Ihre Software in Zukunft läuft, kommen sie ohne Betriebssystem nicht mehr weit.

Für die Darstellung des SuperHires-Modus ist Bit 6 des Registers $0100 zuständig, das zu diesem Zweck gesetzt sein muß. Bisher wurde die Anzahl der Bitplanes über die Bits 12 bis 14 des Registers $0100 angegeben. Damit waren aber nur keine bis sieben Bitplanes möglich. Um acht Planes zu verwenden, muß Bit 4 desselben Registers gesetzt werden. Die Bits 12 bis 14 werden dann nicht mehr beachtet.

Ein größeres Problem bereitet die Angabe der neuen 24-Bit-Farbwerte. Auch bei AA werden nur 12-Bit-Werte von den Registern $0180 bis $01be verarbeitet, und zwar die oberen vier Bit des jeweiligen 8-Bit-RGB-Werts. Damit sind die Register noch 100% kompatibel zum ECS. Um die vollen 24 Bit zu setzen, braucht man das Bit 9 des Registers $0106. Wenn es gesetzt ist, werden die angegebenen Farbwerte als die unteren vier Bits verwendet, sonst als die oberen vier. Beispiel: Setzen der Hintergrundfarbe mit dem RGB-Wert $3f4cd9. Erst müssen die High-, dann die Low-Bits gesetzt werden, also:

```
$01060000
$0180034d
$01060200
$01800fc9
```

(Wer nur 12-Bit Farbwerte verwendet, braucht sich um Bit 9 keine Sorgen machen, denn dieses wird nach jedem copjmp auf 0 gesetzt, er kann wie bisher die Farben setzen)

Ein ähnliches Problem wie bei den Farbwerten tritt durch die neuen, 256 Farben großen Paletten auf, denn es gibt weiterhin nur 32 Farbregister ($0180-$01be). Daher gibt es ab jetzt acht verschiedene 32-Farb-Paletten, die mit Hilfe der Bits 12 bis 14 des Registers $0106 ausgewählt werden. Der durch diese Bits definierte 3-Bit-Wert gibt die Nummer der Palette an (0 bis 7). Beispiel: Es soll die Farbe 177 auf $7a63f9 gesetzt werden. Dafür müssen wir das Register $01a2 der Palette 5 ändern:

```
$01065000
$0180076f
$01065200
$01800a39
```

Um die Auflösung von Sprites zu ändern, braucht man die Bits 7 und 6 von Register $0106. Dabei ist %00 und %01 LoRes, %10 HiRes und %11 SuperHiRes.

Die Breite wird dagegen mit den Bits 3 und 2 im Register $01fc angegeben: %00 für 16 Pixel, %01 und %10 für 32 Pixel und %11 für 64 Pixel Breite. Dabei muß beachtet werden, daß je nach Auflösung die Spriteliste anders gelesen wird: bei 16 Pixeln Word-Weise, d.h. das erste Word ist wie bisher Kontrollregister 1 (C1), das 2.Word C2. Gleiches gilt für die dann folgenden Pointerdaten. Bei 32 Pixeln wird alles als Longword gelesen, also auch die Kontrollregister C1 und C2, die nun jeweils ein Longword sind und bei den 64 Pixel Breiten Sprites müssen die Kontrollregister und die Daten jeweils zwei Longwords groß sein.

Schließlich kann man noch die Farbpalette der Sprites frei wählen. Dies wird durch die Bits 4-7 im Register $010c erreicht. Die damit möglichen Werte zwischen 0 und 15 und geben die Nummer der jeweiligen 16-Farb-Palette an, d.h. die 32-Farb-Paletten werden nochmals aufgeteilt und ermöglichen es somit, daß Spritepaletten nicht erst mit Farbe 16, sondern schon mit Farbe 0 beginnen können.

Geht es oder geht es nicht ?

Nachdem wir jetzt genug über die Programmierung des A1200 geredet haben, gehen wir nun etwas auf seine Bedienung ein. Benutzer ohne Festplatte werden nur sehr schwer die Möglichkeiten ausnutzen können, die ihnen ihr Computer bietet. Wie soll man auch eine Workbench mit 5 Disketten benutzen, wenn man nur ein oder zwei Laufwerke hat? Ganz zu schweigen von Programmen, die unbedingt dieses oder jenes von der Workbench benötigen und nicht separat laufen. Doch auch diejenigen, die ihren Amiga als Spielcomputer ansehen, werden an einer Festplatte kaum vorbeikommen. Inzwischen werden die Spiele immer länger, bieten mehr und bessere Grafik und Sound. Dies hat nur noch schlecht auf zwei oder drei Disketten Platz. Man denke nur an Indiana Jones IV oder Monkey Island II. Auf MS-Dosen werden Spiele bereits nach ihrer Länge und nicht nach ihrer Leistung beurteilt, hoffen wir, daß dies nicht auf dem Amiga Einzug hält. Übrigens behebt eine Festplatte eine etwas lästige Angewohnheit des A1200: die langen Wartezeiten bei einem Reset. Bis zu 10 Sekunden kann es dauern, bis ein A1200 ohne Festplatte wieder »zu sich kommt«.

Und wenn man dann endlich alles fertig angeschlossen und die Workbench installiert, die ganzen Hilfsprogramme und Commodities (z.B. MagicMenu, CycleToMenu, WBGauge, KCommodity, usw. ...) eingebunden hat, dann darf man sich nicht wundern, wenn von den einstmals 2 MByte plötzlich nur noch 800 KByte Speicher frei sind. Auf die 256-Farben -Workbench mit eingebundenen digitalisierten Bildern muß man dann erst recht verzichten. Doch auch mit einer Speichererweiterung kann man diesen Wunsch nur schwer realisieren. Selbst ein A4000 kommt bei 265 Farben und Euro72-Modus ins Schwitzen. Zum flüssigen Arbeiten ist eine Workbench mit höchstens 64 Farben zu empfehlen.

Wer dann noch genug Speicher hat, kann versuchen ein Hintergrundbild in der Workbench einzubinden. Mit Hilfe von WBPattern eigentlich kein Problem. Einfach das Bild definieren, am besten noch ein zweites für die Schubladen-Fenster, und schon sollte die Workbench in neuem Glanz erscheinen. Aber irgendwas scheint nicht zu stimmen, im ersten Moment rührt sich nichts, obwohl die Bilder bereits nachgeladen wurden. Und wenn dann endlich das Bild erscheint, sieht es meistens nicht so aus, wie man es z.B. in DeluxePaint gesehen hat. Warum?

Besser Bilder mit 256 Farben

Da die Workbench leider nur mit 256 Farben, und dann auch nur sehr langsam läuft, müssen die Bilder erst einmal umgerechnet werden. So kann man 256-Farben-Bilder, z.B. GIF-Bilder (i.a. auf PCs verwendetes Format), auch mit weniger Farben auf der Workbench direkt verwenden. Die Ergebnisse dieser Umrechnung sind in der Regel befriedigend bis ausreichend. Es ist daher empfehlenswert, die gewünschten Bilder vorher mit Malprogrammen wie DPaint auf weniger Farben umzurechnen.

Der Workbench-Screen gehört aber nicht nur dem System alleine. Da es sich um einen Public-Screen handelt, kann jedes Programm auf ihn zugreifen. Dazu ist seit OS3.0 das Palette-Sharing erlaubt, welches versucht, die Farbwünsche der verschiedenen Programme miteinander zu kombinieren. Wünscht ein Programm einen roten Farbton, kann es sich nicht einfach eine Farbe dafür nehmen und verän-



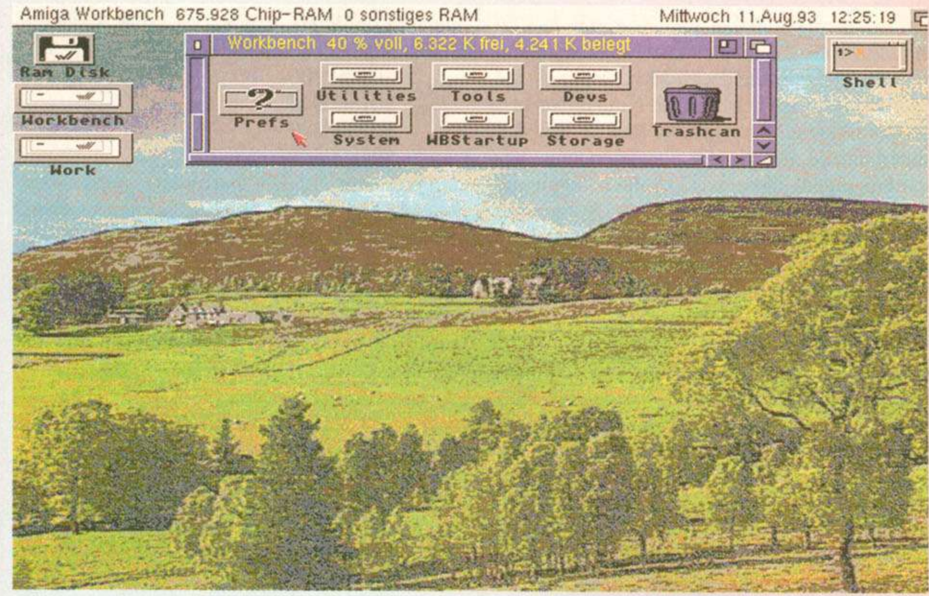
dern, das übernimmt nun das System. Wenn bereits eine ähnliche Farbe vorhanden ist, wird diese dem Programm zur Verfügung gestellt. Aus diesem Grund geht die Workbench etwas »geizig« mit den Farben um und versucht, immer so wenig wie möglich für sich zu belegen. Bilder werden daher auf ein Mindestmaß an Farben reduziert. Selbst wenn man ein 16-Farben-Bild auf einer 64-Farben-Workbench einbindet, wird dieses noch auf weniger Farben reduziert.

Vorsicht vor falschen Farben

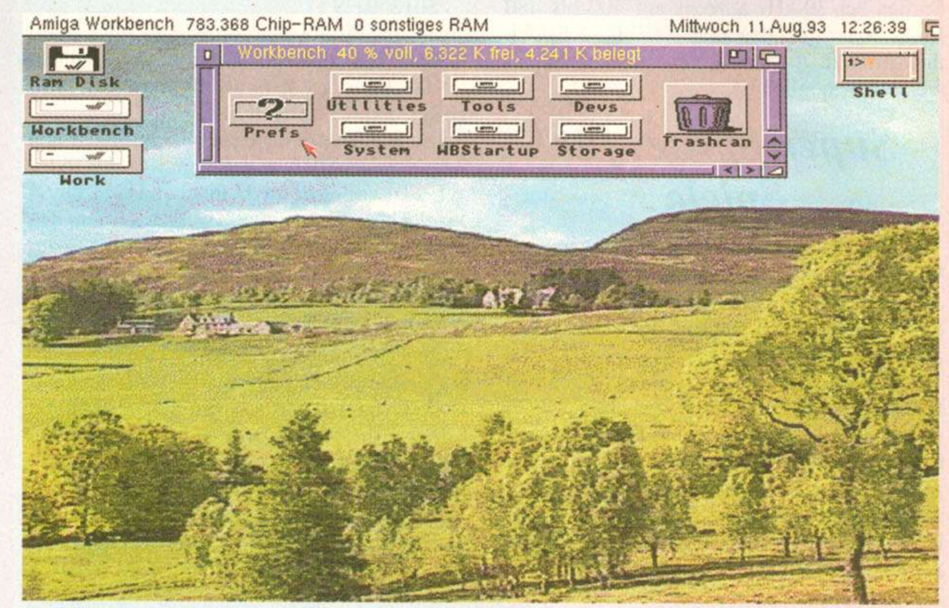
Bei Bildern mit feinen Schattierungen werden durch die Umrechnung viele Farben, die sich untereinander nur leicht unterscheiden, zu einer Farbe zusammengefaßt. Dadurch sehen solche Bilder sehr eintönig und primitiv aus. Bilder mit vielen verschiedenen Farben und hohem Kontrast haben das Problem, daß nicht genug freie Farben zur Verfügung stehen. Daher kann es vorkommen, daß manche Farben durch völlig andere ersetzt werden, z.B. Grün durch Grau. Auf jeden Fall muß man einige Versuche machen, bis ein gutes und annehmbares Ergebnis erzielt wird.

Ein anderes Problem kann die fehlende Uhr darstellen. Wer mit dem Amiga wenig arbeitet und meistens nur spielt, kann darauf sicherlich verzichten. Für die anderen Benutzer ist es schon ein Nachteil. Wer die neuste Kopie eines Textes oder Programms sucht, kann mit der Datumsangabe beim List-Befehl in der Regel wenig anfangen.

Eine mögliche Abhilfe können PD-Programme bringen, die im User-Startup die Zeit mit Hilfe einer Datei weiterzählen. Somit kann man die Datumsangaben wenigstens zum Vergleich heranziehen, welches nun das neuste Projekt ist. Natürlich ergeben sich schnell enorme Abweichungen zur realen Zeit und erfordert immer wieder eine Anpassung. Zeitfanatiker, die die sekundengenaue Zeit auf der Workbench brauchen, werden nicht um den Kauf einer internen Uhr herumkommen, sei es als Einsteckkarte oder als Zusatz bei Speichererweiterungen und Turbokarten.



Bilder auf der Workbench: Im oberen Bild wurde ein 256-Farben Bild direkt vom System umgerechnet, im Bild unten wurde diese Arbeit von DeluxePaint . . .



. . . übernommen und als 32-Farben-Bild gespeichert. Die Unterschiede sind deutlich zu sehen (Workbench läuft mit 64 Farben): Das System verfälscht Farben.

Dateienvielfalt: Chaos oder Ordnung

Da Programme, die das System voll ausnutzen, auf immer mehr Dateien zugreifen, kann dem Benutzer schnell die Übersicht abhanden kommen, was wohin gehört. Es ist keine Seltenheit, wenn zum eigentlichen Programm noch Libraries, Kataloge, Guides, Rexx-Kommandos, Skripte und sonstige Datein kommen. Um das dadurch entstehende Chaos zu umgehen, kann man sich eine Neuerung im Assign-Befehl zu Nutze machen: die »ADD«-Option.

Man legt sich ein zweites »libs«-Verzeichnis an, entweder auf einer anderen Partition als »SYS:« oder mit einem andern Namen. In der Regel gibt es noch eine Partition mit Namen »HD1:«, dort legen wir nun ein Libs-Verzeichnis an und tragen folgede Zeilen im User-Startup ein:

```
C:Assign LIBS:   HD1:libs
C:Assign LIBS:   SYS:libs ADD
```

Nun werden mit »LIBS:« beide Libs-Verzeichnisse angesprochen, neue Libraries werden aber nur in HD1:libs kopiert. Damit hätte man schon mal die System-Libraries von der restlichen Library-Flut getrennt. Man könnte sogar die Libraries auf die Verzeichnisse der dazugehörigen Programme verteilen und mit Hilfe etlicher »Assign ADD«-Anweisungen dafür sorgen, daß das System alle findet.

Das ist aber nicht nur auf das Libs-Verzeichnis beschränkt, es geht mit jedem anderen ASSIGN ebenso. Besonders empfehlenswert ist es, das »C«-Verzeichnis so zu halbieren, denn wenn erst einmal 300 Befehle darin liegen, kann kaum einer mehr sagen, welcher Befehl zum System gehört und welcher nicht.

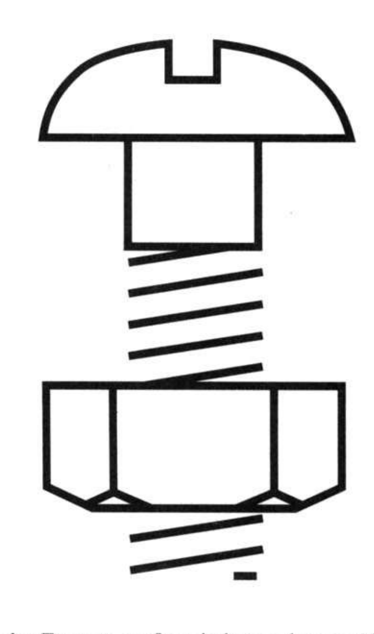
Damit ist unser Überblick über den Amiga 1200 beendet. Wollte man alle Kleinigkeiten erklären, könnte man ein Buch mit 1000 Seiten schreiben, und es wäre noch immer nicht alles erklärt. Doch sollten Ihre Kenntnisse jetzt reichen, um der Leistung des Amiga 1200 würdige Programme zu schreiben. ■

Literatur:

[1] R. Zeitler: »Die goldenen Regeln«, AMIGA-Magazin 2/93
[2] A. Kochann & O. Reiff: »Man spricht Deutsch«, AMIGA-Magazin 10/92-11/92,
[3] A. Kochann & O. Reiff: »Freund & Helfer«, AMIGA-Magazin 4/93,
[4] P. Aurich: »Referenz Shell 3.0«, AMIGA-Magazin 2/93-10/93,
[5] P. Aurich: »Arbeiten mit dem Betriebssystem«, AMIGA-Magazin 4/93,7/93,8/93, M&T Verlag
[6] M. Eckert: »Idealbild«, AMIGA-Magazin 2/93, M&T Verlag
[7] W. Hilf: »Mikroprozessoren für 32-Bit-Systeme«, Band 1, M&T Verlag
[8] Commodore-Amiga, ROM Kernal Reference Manual & Autodocs, Third Edition, Addison Wesley

Installerwerkzeug

Der Software-Installateur



Mit seinem neuen Handwerker, dem »Installer«, setzt Commodore einen neuen Standard, was die Installierung von Software-Paketen auf Festplatte betrifft. Dabei wurde besonderer Wert auf Vielseitigkeit, professionelles Aussehen und leichte Erlernbarkeit gelegt. Mittlerweile gibt´s ihn schon auf Fish-Disk als Public Domain. Wir zeigen, wie Sie diese Vorteile für eigene Projekte nutzen.



von Alexander Kochann

Software-Pakete werden immer umfangreicher: Der Anwender muß neben dem Hauptprogramm zahlreiche Dateien für Text, Grafik oder Sound, Kataloge, ARexx-Scripts, eventuell eine eigene Library und vieles mehr kopieren, um ein Programm überhaupt erst nutzen zu können

Jeder, der bereits mehrere Software-Pakete auf einer Festplatte installiert hat, weiß, daß dies manchmal nicht ganz so einfach ist, wie Hersteller das vorhersagen. Fast jedes Paket hat sein eigenes Installationsprogramm. Und natürlich funktioniert die Installation ausgerechnet auf Ihrem Computer nicht, weil Sie eine andere Festplatte, mehr oder weniger Speicher oder eine Turbokarte besitzen, mit der die mitgelieferte Software nicht zurechtkommt.

Commodore stellt deshalb seit neuem ein Standard-Installationsprogramm, den »Installer«, zur Verfügung. Wer OS3.0 bereits installiert hat, kennt ihn, bzw. das, was man davon sieht. Er läuft aber – laut Commodore – unter allen Versionen des Betriebssystems und auf allen Amiga-Computern.

Um den Installer einsetzen zu können, ist eine Treiberdatei nötig, die man auch Script nennt. Dem erfahrenen Anwender drängt sich da schnell der Vergleich mit ARexx auf, und vielleicht fragt er sich, wie viele Programmiersprachen er noch lernen soll, um als Experte auf dem Gebiet der Amiga-Programmierung zu gelten? Diesem Aspekt hat Commodore Rechnung getragen und deshalb auf leichte Erlernbarkeit geachtet. Der Syntax der Installer-Scripts ist dem der Programmiersprache LISP nachempfunden. Für den Benutzer heißt das einfache Syntax und viele Klammern, denn jeder Befehl wird in runden Klammern eingeschlossen. Eine Ausnahme bilden Kommentarzeilen, die mit einem Semikolon beginnen.

Auch der Umgang mit Variablen ist leicht zu lernen. Sie sind dynamisch, typenlos und müssen nicht vor Gebrauch deklariert werden. Spezielle Richtlinien bei ihrer Benennung gibt es fast keine. Selbstverständlich dürfen Variable keine Zahlen oder Befehle sein. An sonstigen Zeichen sind nur runde Klammern tabu. Zwischen Groß- und Kleinschreibung wird generell nicht unterschieden. Zur besseren Unterscheidung sind jedoch alle Befehle in den Beispielscripts durchgehend groß geschrieben.

Noch bevor wir uns an unser erstes eigenes Script wagen, sollten wir ein Icon dafür erstellen. Der Installer läßt sich zwar auch über die Shell starten, das sollte aber die Ausnahme bleiben. Das Icon muß vom Typ »Projekt« sein und der Installer ist als Standardprogramm einzutragen. Tooltypes, bzw. Merkmale werden mehrere unterstützt (Tabelle 1). Interessant sind für uns im Moment jedoch nur die ersten beiden. Diese sollten immer angegeben werden.

Software einrichten: Alles mit der Maus

Danach kann man dieses Icon mit einem Doppelklick starten. Zuerst wird der Installer nachgeladen, danach das angegebene Script. Dieses wird auf eventuelle syntaktische Fehler überprüft und gestartet oder mit einer Fehlermeldung abgebrochen. Üblicherweise erkundigt sich der Installer jetzt nach Ihrer »Qualifikation«. Sie können sich entweder

– als Einsteiger,
– geübter Benutzer
– oder Experte

zu erkennen geben.

Für den Einsteiger wird das Script fast alles automatisch installieren, dem geübten Einsteiger wird ein gewisses Mitspracherecht erlaubt und der Experte muß zu jeder geplanten Aktion sein Einverständnis geben. Programmieren müssen Sie die Unterschiede nur teilweise selbst. Im Icon können Sie mit

```
MINUSER=EXPERT'
```

diese Abfrage umgehen, da nur noch »Experten« zugelassen werden. Die anderen Qualifikationen heißen auf englisch »novice« und »average«.

Als nächstes kommt auf den Benutzer eine weitere Standardabfrage zu. Er wird gefragt, ob das Programmpaket wirklich oder scheinbar installiert werden soll und ob eine Protokolldatei gewünscht wird? Da eine scheinbare Installation oft überflüssig ist, setzt man am besten

```
PRETEND=FALSE
```

um diesen Punkt zu deaktivieren.

Alle Optionen können Sie auch mit Scripts anderer Anwendungen ausprobieren, doch jetzt wollen wir uns an unser erstes eigenes Script wagen. Es wird nichts weiter tun, als uns zur Installation der Software zu begrüßen und uns von ihrem ordnungsgemäßen Verlauf unterrichten. Immer wenn Sie dem Benutzer eine Nachricht zukommen lassen wollen, können Sie den Befehl »Message« benutzen. Er erwartet als einzigen Parameter einen String als Text oder Variable. Die Endnachricht können Sie mit dem Befehl »Exit« ausgeben, der genau wie »Message« funktioniert und dann die Installation beendet. Mit der Set-Anweisung lassen sich Variablen zuweisen. Mit diesen drei Befehlen können Sie das erste Script bereits verstehen. Wenn Sie es mit mehreren Qualifi-

Unterstützte »Merkmale« im Icon

Merkmal	Bedeutung
SCRIPT	Name und Pfad des Scripts
APPNAME	Name des zu installierenden Projektes
LANGUAGE	Sprache für Installer und Script
DEFUSER	voreingestellte »Qualifikation«
MINUSER	»Mindest-Qualifikation« des Benutzers
LOGFILE	Pfad und Name der Protokolldatei
LOG	Protokolldatei an / aus
PRINT	Druckeroption an / aus
PRETEND	»Scheinbar Installieren« an / aus

kationen ausprobieren, werden Sie feststellen, daß Einsteiger keine Messages bekommen, da bei Ihnen alles automatisch laufen soll und ständiges Bestätigen lästig wäre.

```
; Demo-Script 1
(SET Message1 "Welcome to ...")
(MESSAGE Message1)
(EXIT "See you !")
```

Natürlich hätten wir uns die Variablenzuweisung sparen können. Dennoch ist es sinnvoll, fast alle Texte als Variablen zu vereinbaren. So können Sie auch die Möglichkeit des Installers nutzen, verschiedene Sprachen zu unterstützen. Dazu müssen Sie nur die gewählte Sprache abfragen und die Texte entsprechend setzen. Sie befindet sich in der vordefinierten Variablen »@language« (siehe Tabelle 2). Abgefragt wird das Ganze mit einem If-Block. Dieser hat folgendes Format:

```
(IF (Bedingung) (THEN-Block) (ELSE-Block))
```

Bevor der Installer mit dem Kopieren von Dateien beginnen kann, muß er wissen, wo er sie findet und wo sie später hingehören. Als Beispiel arbeiten wir mit der Diskette »InstallerDemo«, die wir uns zunächst anlegen. Auf ihr legen Sie dann die benötigten Verzeichnisse und Dateien an. Auf den Inhalt der Dateien kommt es im Moment noch nicht an. Als erstes testen wir, ob die Diskette oder ein anderer Datenträger verfügbar ist. Dazu dient ASKDISK. Es erwartet keinen direkten Parameter, sondern zum Teil optionale Standardparameter. Diese werden immer in Klammern angegeben. Mit (DEST <name>) wird der Name der Diskette angegeben und mit (PROMPT <text>) ein Text, der den Benutzer dazu auffordert die Diskette einzulegen, wenn sie noch nicht in einem Laufwerk ist. Der Parameter (HELP <helptext>) gibt wie bei vielen Befehlen eine detailliertere Erläuterung zum geplanten Vorhaben.

Tabelle 2: Vordefinierte Variablen

@abort-button	Text für das 'Abort Install'-Gadget
@app-name	Name der Anwendung
@default-dest	Standard Installations-Pfad
@each-name	Dateiname in der foreach-Schleife
@each-type	Objekttyp in der foreach-Schleife
@error-msg	Standard-Fehlermeldung
@execute-dir	Pfad für weitere Shell-Kommandos
@icon	Pfadname des Script-Icons
@ioerr	Nummer der letzten DOS-Fehlermeldung
@language	voreingestellte Sprache
@pretend	1: scheinbar installieren, 0: wirklich
@special-msg	Fehlermeldung bei fatalen Fehlern
@user-level	0: Einsteiger, 1: geübter Benutzer, 2: Experte
@<function>-help	Standardhilfstexte (siehe Tabelle 3)

Im Gegensatz zum bekannten AmigaBASIC werden die Schlüsselwörter THEN oder ELSE durch runde Klammern ersetzt. Am besten betrachten Sie das zweite Script, das eine verbesserte Version des ersten darstellt.

```
; Demo-Script 2
(IF (= @language "deutsch")
  (
   (SET Message1 "Willkommen bei ...")
   (SET Message2 "Und tschüß !")
  )
  (
   (SET Message1 "Welcome to ...")
   (SET Message2 "See you !")
  )
)

(MESSAGE Message1)
(EXIT Message2)
```

Gewöhnungsbedürftig sind die vorangestellten Vergleichszeichen wie »=«, »>«, »<=« oder »<>« für ungleich. Auch mathematische Funktionen werden so behandelt, wie (SET a (+ a 1)), was a um eins erhöht. Neben Addition und Subtraktion sind auch Multiplikation und Division implementiert, die durch die üblichen Rechenoperatoren angezeigt werden. Die Frage nach unterschiedlichen Prioritäten ist überflüssig, da in einer Klammer immer nur eine Rechenart stehen kann.

An folgenden Zeilen kommt der Benutzer nicht vorbei, ohne die Diskette einzulegen.

```
(ASKDISK (PROMPT "Bitte 'InstallDemo'-Disk
          einlegen !")
   (DEST  "InstallDemo")
   (HELP  "Ich brauche die Disk !"))
```

Als nächstes kümmern wir uns um eine Zielpartition oder -verzeichnis. Dazu dient die Funktion ASKDIR. Auch sie unterstützt die Parameter Prompt und Help. Außerdem kann über DEFAULT eine Voreinstellung angegeben werden. Am besten verwenden Sie jedoch dem Vorschlag des Installers, den Sie in »@default-dest« finden. Dieser Variablen sollte man dann auch den neuen Wert zuweisen.

```
(SET @default-dest
  (ASKDIR (PROMPT  "Bitte Ziel aussuchen:")
      (HELP    "Ich brauche ein Ziel !")
      (DEFAULT @default-dest)))
```

Jetzt können wir uns schon an das einfache Kopieren von Dateien wagen. Dazu gibt es den Befehl COPYFILES. Im einfachsten Fall werden Quelle und Ziel benötigt.

```
(COPYFILES (SOURCE "InstallDemo:testfile")
   (DEST   @default-dest))
```

Wenn Sie jetzt die letzten drei Beispiele zusammenfügen, haben Sie bereits ein durchaus sinnvolles Script, denn mit diesen wenigen Befehlen könnten Sie bereits alle Dateien kopieren, die benötigt werden. Aber der Installer stellt noch wesentlich umfangreichere und mächtigere Funktionen zur Verfügung, vor allem wenn es darum geht, mit dem Benutzer zu kommunizieren. Dazu gibt es neben AskDisk und AskDir noch andere Abfragen, die ganz ähnlich arbeiten.

Mit AskFile, AskString und AskNumber fragen Sie einzelne Dateien, Texte und Zahlen ab. Alle erfordern Prompt und Help. Da es für den Programmierer sehr aufwendig wäre, für jede Funktion einen eigenen Helptext zu schreiben, sind zu den in Tabelle 3 aufgeführten Befehlen und Funktionen bereits Standardtexte definiert. Weitere Fragetypen werden durch AskBool, AskChoice und AskOptions abgedeckt. Bei den letzteren muß der Parameter CHOICES verwendet werden. Er enthält nacheinander die Alternativen als einzelne Strings. Dies eignet sich hervorragend, um beispielsweise die gewünschten Sprachen abzufragen, die ein Programm unterstützen kann. Sogar eine Vorauswahl kann mit Hilfe von DEFAULT getroffen werden. Ihr wird einfach die Nummer des Strings übergeben, wobei die Zählung bei eins anfängt.

```
(IF (ASKBOOL (PROMPT "Sprachen install.?")
        (HELP   @ASKBOOL-HELP))
  (SET sprachen
  (ASKOPTIONS (PROMPT  "Sprachen wählen :")
       (HELP    "Welche Sprachen...")
       (CHOICES "deutsch"
                "français"
                "italiano")
       (DEFAULT 1))))
```

Die sichtbare Abfrage wäre mit diesem Programmteil erledigt. Was aber enthält die Variable Sprachen jetzt und wie kann man die einzelnen Möglichkeiten wieder abfragen? Eine einfache Möglichkeit ist eine If-Abfrage in Verbindung mit IN und CopyFiles.

```
(IF (IN sprachen 1)
  (COPYFILES (SOURCE "InstallDemo:locale/
          catalogs/deutsch/test.catalog")
     (DEST   "locale:catalogs/deutsch")))
```

Allerdings wird für jede Sprache eine eigene Abfrage mit IN benötigt, die übrigens bitweise arbeitet. Kompakter geht dies mit einer WHILE-Schleife, auf die wir später noch zu sprechen kommen, weil sie noch einiges Wissen erfordert und deshalb erst gegen Ende besprochen wird.

Gerade wenn man auf allgemeinere Ressourcen wie Kataloge oder Libraries zugreifen will, ist es wichtig zu testen, ob sie bereits existieren und welche Versionsnummer sie haben. Ist die Verisonsnummer kleiner als die unserer Datei, ist ein Kopieren überflüssig, oft sogar schädlich für Programme, die bereits auf diese Ressource zugreifen.

```
(IF (EXISTS("C:copy"))
 ((SET version (GETVERSION("C:copy")))
 (IF (< version (GETVERSION(
             "InstallDemo:c/copy")))
  (COPYFILES (SOURCE "InstallDemo:c/copy")
      (DEST   "C:"))
  (message "Copy wird nicht kopiert !"))
 )
 (COPYFILES (SOURCE "InstallDemo:c/copy")
     (DEST   "C:")))
```

Mit EXISTS können Sie nicht nur erkennen, ob eine Datei existiert, sondern auch, ob es sich um eine Datei oder ein Verzeichnis handelt. Ist die Datei nicht vorhanden, wird Null zurückgegeben. Wollen Sie testen, ob sich die Datei auf einem Datenträger befindet, der gerade nicht erreichbar ist, geben Sie (NOREQ) an, damit keine Eingabeaufforderung erscheint. Wird das »File« gefunden, wird Eins für Datei oder Zwei für Verzeichnis zurückgegeben. Mit GETVERSION lesen Sie die Versionsnummer der Datei: eine 32-Bit-Zahl, deren oberes Word die Version und unteres Word die Revision enthält. Benötigen Sie beide getrennt, teilen Sie die Version durch 65536. Der ganzzahlige Rest ist die Revision. Getversion ohne Parameter liefert die Kickstartversion.

Wie Sie im Script sehen, taucht die Anweisung »Copyfiles« zweimal auf. Werden Befehlsfolgen noch umfangreicher und tauchen häufiger auf, ist es sinnvoll, diese in Prozeduren zusammenzufassen. Das ist in diesem Fall recht einfach, da noch keine Variablen übergeben werden können und auch lokale Variablen nicht definierbar sind. Die Deklaration besteht also nur aus dem Namen und der Befehlsfolge.

Noch einfacher ist der Aufruf, der einfach durch den Namen der Prozedur in Klammern angezeigt wird. Das nächste Script zeigt die Deklaration und einen Aufruf.

```
(PROCEDURE CopyCopy
 (COPYFILES (SOURCE "InstallDemo:c/copy")
     (DEST   "C:")) ) (CopyCopy)
```

Tabelle 3: Funktionen mit Standard-Hilfstexten

askchoice	askdir	askdisk
askfile	asknumber	askoptions
askstring	copyfiles	copylib
makedir	startup	

Tabelle 4: Standardparameter

Standardparameter	Beschreibung
ALL	alle Dateien kopieren
APPEND <string>	Text in eine ASCII-Datei einfügen
ASSIGNS	Gibt ein Assign als Diskettenlaufwerk an (nur zum Debuggen verwenden !)
CHOICES <string1> <string2>	Auswahlmenütexte setzen
COMMAND <string1> <string2>...	Scriptbefehle für s:user-startup
CONFIRM <user-level>	welche Benutzer die Aktion bestätigen sollen
DEFAULT <value>	Voreinstellung setzen
DELOPTS <opion1> <option2> ...	Löscht die angegebenen Optionen (s. OPTIONAL)
DEST <file>	Ausgabedatei, bzw. -verzeichnis festlegen
DISK	Namen von Laufwerke vor Assigns anzeigen
FILES	keine Unterverzeichnisse kopieren
FONTS	Dateien mit der Endung '.font' werden nicht angezeigt, aber kopiert
HELP <string1> <string2> ...	auf Wunsch Hilfstext anzeigen
INCLUDE <file>	ASCII-Datei in eine andere einfügen
INFOS	auch Piktogramme mitkopieren
NEWNAME <name>	neuer Datei- oder Diskettennamen
NOGAUGE	kein Füllbalken beim Kopieren
NOPOSITION	Position des Piktogramms freigeben
NOREQ	keinen Requester darstellen
OPTIONAL <option1> <option2> ...	Setzt folgende Optionen: "fail" - Abbruch bei Fehler "nofail" - kein Abbruch bei Fehler "oknodelete" - kein Abbruch bei schreibgeschützten Dateien "force" - Protection-Bits nicht beachten "askuser" - Benutzer befragen, wenn Datei schreibgeschützt ist
PATTERN <string>	Patternmatching beachten
PROMPT <string1> <string2> ...	Info-Text zu einem Befehl
QUIET	Installer ohne Nachricht beenden
RANGE <min> <max>	Reichweite für Zahleneingaben
SAFE	auch bei 'Scheinbar Installieren' durchführen
SETTOOLTYPE <tooltype> <value>	Merkmal im Piktogramm setzen
SETDEFAULTTOOL <value>	Standardprogramm im Piktogramm setzen
SETSTACK <value>	Stackgröße im Piktogramm setzen
SOURCE <file>	Eingabedatei, bzw. -verzeichnis
SWAPCOLORS	Farben des Piktogramms ändern (für OS1.3-Icons)

Am einfachesten geht das Kopieren allerdings mit dem COPYLIB-Befehl. Dieser kopiert nicht etwa nur Libraries, sondern beliebige Dateien, die einen Versionstring enthalten müssen, der sich nach dem offiziellen Standard von Commodore richtet. Benutzt wird er ganz ähnlich wie CopyFiles. Dabei wird jedoch automatisch die Versionsnummer beachtet. Ein anderes Problem, das beim Updaten von Software auftritt, ist, daß überflüssige Dateien gelöscht werden sollten. Dazu gibt es generell zwei Möglichkeiten, die elegantere über DELETE, dem der Dateiname und Parameter wie help und prompt übergeben werden. Über (OPTIONAL <option>) können noch zusätzliche Sicherungen eingebaut werden. »FORCE« löscht auch geschützte Dateien und »ASKUSER« erbittet eine Bestätigung vom geübteren Benutzer in einem solchen Fall.

Die zweite Möglichkeit geht den Weg über die Shell-Befehle. Diese können alle als externe Kommandos verwendet werden. Dazu dienen die Kommandos RUN und EXECUTE. Der Unterschied besteht nicht etwa darin, daß Run im Hintergrund läuft, während Execute solange wartet, bis der Befehl ausgeführt wurde, sondern darin, daß mit Run ausführbare Programme gestartet werden, während Execute ganze Shell-Scripts lädt und abarbeitet. Natürlich sollte auch die dritte Art externer Kommandos nicht fehlen: Mit REXX können sogar ARexx-Scripts eingebunden werden. Da diesen Befehlen jeweils nur die Script-Datei, bzw. das Programm mit einer eventuellen Kommandozeile übergeben wird, wollen wir sie hier nicht näher behandeln.

Installer mit mächtigen Befehlen

Nicht nur durch diese externen Kommandos wird der Installer so vielseitig, daß fast keine internen Befehle mehr notwendig wären. Dennoch stellt der Installer auch Kommandos wie Rename, Protect, MakeDir u.ä. zur Verfügung, was die Fehlerabfrage enorm erleichtert im Gegensatz zu den externen Befehlen. Auch diese Befehle benutzen fast nur Standardparameter, so daß sie leicht zu erlernen sind.

Die großen Vorteile des Installers liegen zusätzlich in drei weiteren Bereichen: der Änderung von Textdateien, wie der Startup-Sequence, sowie der Manipulation von Piktogrammen und den zahlreichen Funktionen zur Verwaltung von Strings, was über externe Befehle allein wohl schlecht möglich wäre.

Beginnen wir mit der Stringverwaltung, da diese oft Grundlage für die beiden anderen Gebiete ist. Strings werden bekanntlich von Anführungszeichen umgeben. Dadurch entsteht das erste Problem: wie kann ich Anführungszeichen in Strings einfügen. Da das Problem nicht neu ist, ist auch die Lösung nicht die neueste. Wie in C üblich werden Sonderzeichen durch Schrägstriche eingeleitet:

– \" fügt ein Paar Anführungszeichen ein,
– \n ein LineFeed,
– \r einen Returncode,

– \t einen Tabulator
– und \0 schließlich ein Nullbyte.

Um einen Schrägstrich auszugeben, muß man dann allerdings gleich zwei tippen, damit Verwechslungen vermieden werden.

Die einfachste Funktion im Bereich der Zeichenketten ist das Aneinanderketten zweier oder mehrerer Strings. Dies übernimmt die Funktion CAT. Sie hat ihren Namen nicht etwa von einer Katze, sondern von dem englischen »concatenation« – Verkettung.

Zauberei mit Ziffern und Zeichen

Damit die C-Programmierer auch nicht auf ihre printf-Routine verzichten müssen, wurde auch diese von den Programmierern des Installers umgesetzt. Diese Funktion hat keinen eigentlichen Namen, sondern enthält in Klammern nur den Formatstring, gefolgt von den Argumenten. So ist es kein Problem mehr, variable Zahlen und Texte in einer Zeichenkette unterzubringen. Zu beachten ist nur, daß alle Zahlen 32-Bit breit sind. Das Beispiel zeigt außerdem, wie man dem Benutzer im Fensterrahmen Auskunft über den Stand der Dinge geben kann.

```
(SET prozente 66)
(COMPLETE prozente)
(MESSAGE ("Bereits %ld Prozent
          installiert !" prozente))
```

Benötigt man die Länge eines Strings, kann man sich diese mit STRLEN besorgen. Auch das Zerlegen von Zeichenketten ist kein großes Problem mit dem Installer. Am einfachsten geht es mit SUBSTR. Neben dem String wird die Position des ersten Zeichens angegeben, das übernommen werden soll. Die Länge des Teilstrings ist optional. Wird sie weggelassen, wird der String bis zum Ende kopiert.

Da der Installer hauptsächlich mit Dateien und deren Namen arbeitet, war es dringend notwendig, auch dafür spezielle Stringfunktionen zu implementieren. (FILEONLY <path>) gibt zum Beispiel nur den Dateinamen ohne den Pfad zurück, während PATHONLY eben diesen ohne Dateinamen liefert.

Und nicht nur das Zerlegen von Pfaden ist möglich. Mit dem Aufruf(TACKON <path> <file>) können Dateinamen wieder korrekt zusammengesetzt werden, ohne daß man sich um Doppelpunkte oder Schrägstriche kümmern müßte. Und auch eine weitere Funktion beschäftigt sich mit Pfaden, nämlich EXPANDPATH. Jegliche Assigns werden darin aufgelöst, so daß nur der Pfad über physikalische Geräte, bzw. Schubladen übrigbleibt. Zum Abschluß dieser Funktionsgruppe noch ein kleines Bonbon zum Patternmatching, das seit OS2.0 auch in der dos.library erheblich an Bedeutung gewonnen hat.

```
(SET file "x.info")
(IF (PATMATCH "#?.info" file)
   (MESSAGE "Ist Piktogramm !")
   (MESSAGE "Ist kein Piktogramm !"))
```

Auf diese Art ist ganz einfach feststellbar, ob eine Datei eine bestimmte Endung hat, wie Kataloge oder Piktogramme. Gerade letztere sind für die Installation von Software oft zwingend notwendig und müssen ziemlich häufig an die Bedürfnisse des Benutzers angepaßt werden. Dafür existiert ein einziger aber sehr wirkungsvoller und mächtiger Befehl: TOOLTYPE. Ihm können neben Prompt, Help und Dest auch noch die zu ändernden Werte wie Stan-

Tabelle 5: Befehle im Überblick

Befehl mit Parametern	Beschreibung
ABORT <string1> <string2> ...	Befehl führt zum Abbruch des Installers und gibt vorher die angegebenen Texte aus.
COMPLETE <num>	Titel des Fensters, der angibt, wieviel Prozent bereits installiert wurden, wird auf den neuen Wert gesetzt.
COPYFILES prompt help source dest newname choices pattern confirm all files infos safe nogauge optional delopts fonts	Datei(en) aus dem Quellverzeichnis ins Ziel kopieren. Dabei wird keine Versionsnummer beachtet. Unterverzeichnisse und Piktogramme können optional mitkopiert werden.
COPYLIB prompt help source dest newname choices pattern confirm all files infos safe nogauge optional delopts	Kopiert nur Dateien, in die ein Standard-Versionstring implemetiert ist. Andere Dateien werden nicht kopiert. Ist die Datei bereits in einer höheren Version vorhanden, wird sie nicht durch die ältere Version überschrieben.
DEBUG <parameters> ...	Angegebene Parameter werden in der Shell ausgegeben. Der Befehl sollte nur zum Debuggen verwendet werden.
DELETE <file> help prompt confirm safe optional delopts	Die angegebene Datei wird normalerweise gelöscht. Optional kann der Benutzer gefragt werden, ob er das wirklich wünscht.
EXECUTE <script> help prompt confirm safe	Das angegebene Shell-Script wird ausgeführt.
EXIT <string1> <string2> ... quiet	Installer endet still oder mit dem angegebenen Text.
FOREACH <dir> <pattern> <statements>	Die in <statements> definierten Befehle werden auf alle passenden Dateien des Verzeichnisses angewandt.
IF <expression> <then-statements> <else-statements>	Ist der Ausdruck <expression> wahr, so werden die in <then-statements> definierten Befehle ausgeführt, im anderen Fall die <else-statements>.
MAKEASSIGN <assign> [<path>] safe	Wie der Shell-Befehl Assign. Beim <assign> wird jedoch kein Doppelpunkt angehängt. Fehlt ein Pfad, wird das Assign entfernt.
MAKEDIR <name> prompt help infos confirm safe	Wie der Shell-Befehl MakeDir.
MESSAGE <string1> <string2> ... help	Ausgabe einer Nachricht für den geübteren Benutzer
ONERROR <statements>	<statements> gibt Befehle im Fehlerfall an.
PROCEDURE <name> <statements>	Faßt Befehle in <statements> als Prozedur »<name>« zusammen.
PROTECT <file> [<dosmask>] [<decimal mask] safe	Setzt die Protection-Bits einer Datei, oder liest sie aus, wenn keine Maske angegeben ist.
RENAME <oldname> <newname> help prompt confirm safe disk	Wie der Shell-Befehl Rename. disk bedeutet, daß eine Diskette umbenannt werden soll.
REXX <script> help prompt confirm safe	Das angegebene ARexx-Script wird ausgeführt.
RUN <program> help prompt confirm safe	Das angegebene Programm oder Shell-Kommando wird ausgeführt.
SET <var> <expression> <var2> <expression2> ...	Setzt die Variable <var> auf den Rückgabewert des Ausdrucks <expression>. Mehrfache Zuweisungen sind möglich.
STARTUP <appname> command prompt help	Die in »command« definierten Textzeilen werden in die Datei s:startup-sequence, bzw. s:user-startup eingefügt. <appname> ist dabei der Name der Anwendung.
TEXTFILE prompt help dest append include confirm safe	Eine ASCII-Datei wird je nach Parametern erzeugt.
TOOLTYPE prompt help dest confirm noposition safe setstack settooltype setdefaulttool swapcolors	Die Daten eines Piktogramms werden geändert.
TRAP <flags> <statements>	Bei einem bestimmten Ereignis sollen die angegebenen Befehle ausgeführt werden. Die Ereignisse sind: 1: ABORT, 2: NoMem, 3: Error, 4: DosError, 5: BadArgs
UNTIL <expression> <statements>	Die angegebenen Befehle werden ausgeführt, solange der Ausdruck <expression> falsch ist.
USER <level>	Setzt die Benutzerqualifikation neu. Sollte nur zum Debuggen von Scripts benutzt werden und nicht in fertigen Scripts
WELCOME <string1> <string2> ...	Ersetzt die Standard-Begrüßung
WHILE <expression> <statements>	Solange der angegebene Ausdruck wahr ist, werden die Befehle ausgeführt.
WORKING <string1> <string2> ...	Zeigt dem Benutzer an, daß der Installer arbeitet.

Tabelle 6: Funktionen im Überblick

Funktion mit Parametern	Beschreibung
= <expression1> <expression2>	Vergleicht zwei Ausdrücke und gibt TRUE oder FALSE zurück.
<> <expression1> <expression2>	»ungleich
> <expression1> <expression2>	»größer als«;
>= <expression1> <expression2>	»größer als oder gleich«
< <expression1> <expression2>	»kleiner als«
<= <expression1> <expression2>	»kleiner als oder gleich«
+ <expression1> <expression2>	Gibt die Summe beider Ausdrücke zurück.
- <expression1> <expression2>	Gibt die Differenz beider Ausdrücke zurück.
* <expression1> <expression2>	Gibt das Produkt beider Ausdrücke zurück.
/ <expression1> <expression2>	Gibt den Quotienten beider Ausdrücke zurück.
AND <expression1> <expression2>	Logisches AND
OR <expression1> <expression2>	Logisches OR
XOR <expression1> <expression2>	Logisches XOR
NOT <expression1> <expression2>	Logisches NOT
BITAND <expression1> <expression2>	Binäres AND
BITOR <expression1> <expression2>	Binäres OR
BITXOR <expression1> <expression2>	Binäres XOR
BITNOT <expression1> <expression2>	Binäres NOT
SHIFTLEFT <value> <bits>	Schiebt <value> um <bits> Bits nach links.
SHIFTRIGHT <value> <bits>	Schiebt <value> um <bits> Bits nach rechts.
IN <expression> <bitnumber1> ...	Logisches AND aus <expression> und den angegebenen Bits.
<format> <arg1> <arg2> ...	Formatierung wie printf, bzw. RawDoFmt().
ASKDIR prompt help default newpath disk	Stellt eine Auswahlbox dar, aus der nur Verzeichnisse, Disketten oder Assigns gewählt werden können.
ASKFILE prompt help default newpath disk	Stellt eine Datei-Auswahlbox dar.
ASKSTRING prompt help default	Ein Stringgadget wird dargestellt und abgefragt.
ASKNUMBER prompt help range default	Wie ASKSTRING, es sind jedoch nur Zahlen erlaubt.
ASKCHOICE prompt help choices default	Stellt die in CHOICES angegebenen Auswahlmöglichkeiten dar, aus denen der Benutzer genau eine wählen kann.
ASKOPTIONS prompt help choices default	Stellt die in CHOICES angegebenen Auswahlmöglichkeiten dar, aus denen der Benutzer beliebig viele auswählen kann.
ASKBOOL prompt help default choices	Ja-Nein-Abfrage (0: Nein, 1: Ja)
ASKDISK prompt help dest newname assigns	Test, ob ein Speichermedium verfügbar ist. Falls nicht, wird der Benutzer aufgefordert, es verfügbar zu machen.
CAT <string1> <string2> ...	Verknüpft mehrere Strings miteinander.
DATABASE <feature>	Gibt Daten über den Amiga zurück. "vblank" - z.B. "50" oder "60" "cpu" - z.B. "68000" oder "68040" "total-mem" - Anzahl aller freien Bytes als String "graphics-mem" - freie Bytes im CHIP-Memory als String "chiprev" - z.B. "AGNUS", "ECS" oder "AA"
EXISTS <filename> noreq	Testet den Type eines Files. 0: nicht vorhanden 1: Datei 2: Verzeichnis
EXPANDPATH <path>	Entfernt Assigns aus dem Pfadnamen und ersetzt sie durch den physikalischen Pfad (Diskettennamen und Unterverzeichnisse).
EARLIER <file1> <file2>	TRUE, wenn <file1> älter als <file2> ist.
FILEONLY <path>	Schneidet von einem Dateinamen die Pfadangaben ab, so daß nur noch der eigentliche Name übrigbleibt.
GETASSIGN <name> <options>	Übergibt den Pfadnamen eines Assigns. Options: 'v': Volumes, 'a': Logische Geräte, 'd': Devices
GETDEVICE <path>	Gibt den übergeordneten Pfad eines Assigns an.
GETDISKSPACE <path>	Übergibt die Anzahl der freien Bytes auf diesem Datenträger oder -1, wenn der Pfad ungültig war.
GETENV <name>	Liest den Wert einer Environment-Variablen
GETSIZE <file>	Liefert die Größe einer Datei in Bytes.
GETSUM <file>	Berechnet die Checksumme einer Datei, wie sie vom Programm »InstSum« berechnet wird.
GETVERSION <file> resident	Durchsucht Datei oder residentes Modul nach Versionsnummer. Diese wird mit der Revision als 32-Bit-Integer zurückgegeben. Ohne Parameter wird die Kickstartversion zurückgegeben.
PATHONLY <path>	Schneidet von einem Dateinamen den eigentlichen Dateinamen ab, so daß nur noch der Pfad übrigbleibt.
PATMATCH <pattern> <string>	TRUE, wenn <string> in das angegebene Muster paßt.
SELECT <n> <item1> <item2> ...	Gibt <item n> oder den letzten Eintrag zurück, wenn weniger als n Einträge definiert wurden.
STRLEN <string>	Länge des Strings ohne Nullbyte
SUBSTR <string> <start> [<count>]	Liest ab der Position <start> alle Bytes aus einem String aus. Wird <count> angegeben, so werden nur soviele Bytes gelesen.
TRANSCRIPT <string1> <string2> ...	Schreibt die angegebenen Texte ins Logfile.
TACKON <path> <file>	Der Dateiname <file> wird an einen vorhandenen Pfad angefügt.

dardprogramm, Merkmale oder Stackgröße übergeben werden. Dadurch wird nur einmal auf eine Piktogramm-Datei zugegriffen und alle Werte mit einem Schlag verändert. Genau dies zeigt unser nächstes Script.

```
(TOOLTYPE (DEST "ram:Install_Demo")
      (SETTOOLTYPE "X" "20")
      (SETTOOLTYPE "Y" "10")
      (SETDEFAULTTOOL "DefaultTool")
      (SETSTACK 10000)
      (NOPOSITION))
```

Nicht nur Piktogramme, auch die Startup-Dateien müssen immer wieder den Wünschen der Benutzer angepaßt werden, um beispielsweise Assigns korrekt zu setzen, Zugriffspfade zu erweitern oder Programmodule zu laden. Mit STARTUP wird der User-Startup um Textzeilen erweitert. Existiert diese Datei nicht, wird die Startup-Sequence geändert. Neben den bekannten Parametern Prompt und Help wird StartUp noch der Name des Projekts als String übergeben. Die Zeilen, die eingefügt werden sollen, stehen hinter dem Schlüsselwort COMMAND.

```
(STARTUP " Assigns für ..."
      (PROMPT  "Modifiziere StartUp !")
      (HELP    "Blabla")
      (COMMAND "Assign X: Work:x"))
```

Ganz ähnlich können Sie auch beliebige andere Textdateien anlegen. Der Befehl dazu heißt sinnigerweise TEXTFILE und verfügt über ähnliche Parameter wie Startup. Mit (INCLUDE <file>) kann eine andere Textdatei in die bestehende eingefügt werden, (APPEND <string>) fügt beliebige Zeichenketten hinzu. Dieser Befehl ist dazu gedacht, Batch-Dateien oder andere Kommandofolgen wie ARexx-Scripts anzulegen, deren Inhalt teilweise variabel ist, weil er sich z.B. nach dem Installationspfad richtet.

Die eigentlichen Befehle und Funktionen, abgesehen von binären und logischen Operatoren, kennen Sie nun. Gegen Ende möchten wir Sie noch mit zwei nützlichen Strukturierungen vertraut machen: die WHILE- und die REPEAT-UNTIL-Schleifen. Von der Bedeutung dürften sie allen Programmierern bekannt sein. Ihr einziger Unterschied besteht darin, daß bei einer While-Schleife die Bedingung zu Beginn abgefragt wird. Dies geschieht bei der gleichen Konstruktion mit Repeat erst am Ende einer jeden Schleife, so daß sie mindestens einmal durchlaufen wird.

Vom Syntax sind beide nicht besonders schwierig, wie wir das von allen anderen Installer-Befehlen ja schon kennen. Ähnlich wie bei der If-Abfrage kommt zuerst das Schlüsselwort While, dann in Klammern die Bedingung und in einem weiteren Klammernpaar die auszuführenden Befehle. Die Repeat-Until-Schleife ist vom Syntax her identisch. Das Schlüsselwort heißt jedoch nicht Repeat, sondern Until. Das nächste Script zählt die Variable i von 0 bis 10 hoch und dann wieder abwärts bis 0.

```
(WHILE (< i 10)
    (SET i (+ i 1)))
(UNTIL (= i 0)
    (SET i (- i 1)))
```

OS2.1 & 3.0

In solchen Schleifen ist auch die SELECT-Anweisung von großem Vorteil. Gerade im Zusammenhang mit der Auswertung von Abfragen über AskOptions oder AskChoice. Gibt man folgende fünf optionalen ARexx-Scripte als Wahl einer Abfrage an, kann man sie über Select sehr gut wieder abfragen. Select benötigt eine Variable, nach deren Wert die Auswahl getroffen werden soll. Diese ist in unserem Beispiel die Laufvariable n. Ihr folgen die Auswahlmöglichkeiten für den Fall, daß die Variable 0, 1, 2, 3 oder 4 ist. Die letzte Angabe ist für den Fall gedacht, daß keine der anderen Möglichkeiten zutrifft. Neu ist außerdem die Verwendung eines Patterns bei CopyFiles. Es besteht aus (%<dospattern>|<dospattern>|...).

```
(SET rexxs
 (ASKOPTIONS
  (PROMPT  "ARexx-Scripte wählen")
  (HELP    "blabla")
  (CHOICES "Start.rexx"
           "End.rexx"
           "Pause.rexx"
           "Break.rexx"
           "Repeat.rexx")))
(SET filepat "(%")
(SET n 0)
(WHILE
 (SET file
  (SELECT n "Start.rexx"
           "End.rexx"
           "Pause.rexx"
           "Break.rexx"
           "Repeat.rexx"
           ""))
 (IF (IN rexxs n)
  (SET filepat (CAT filepat "|" file)))
 (SET n (+ n 1)))
(SET filepat (CAT filepat ")"))
(COPYFILES (SOURCE "InstallDemo:Rexx")
     (DEST    (TACKON @default-dest "Rexx"))
     (PATTERN filepat))
```

Der Installer kennt noch eine dritte Art von Schleifen. Mit FOREACH können Sie für jeden Eintrag eines Verzeichnisses bestimmte Befehle ausführen lassen. Dies ist vor allem durch die direkte Unterstützung eines Pattern sehr komfortabel. Angenommen Ihr Hauptprogramm benötigt für unterschiedliche Betriebssystemversionen verschiedene Dateien, wie Libraries. Dann können Sie die Dateien im gleichen Verzeichnis ablegen, aber je nach Endung kopieren. Die internen Variablen @each-name und @each-type enthalten dabei den Dateinamen, bzw. den Objekttypen der Datei, wie er auch im FileInfoBlock bei fib_DirEntryType definiert ist. Unser Beispiel kopiert alle Dateien mit der Endung »_1.3« in das angegebene Verzeichnis.

```
(SET mypattern "#?_1.3")
(FOREACH "InstallDemo:" mypattern
 (COPYFILES (SOURCE (
       TACKON "InstallDemo:" @each-name))
    (DEST "ram:")))
```

Nachdem Sie jetzt fast alle Befehle, Funktionen und deren Parameter kennen, müßten Sie eigentlich in der Lage sein, eigene Scripts für den Installer zu schreiben. Dazu noch ein paar nützliche Tips: Probieren Sie ein Script mit jeder Benutzer-Qualifikation aus, damit es nicht zu unliebsamen Effekten bei einem der drei Grade kommt. Gewöhnen Sie sich außerdem an, die Klammern sinnvoll und übersichtlich zu setzen. C-Programmierern sollte das etwas leichter fallen, als ihren Assembler-Kollegen.

Sollte dennoch ein Fehler einmal nicht auffindbar sein, was bei den spartanischen Fehlermeldungen des Installers nicht weiter verwunderlich ist, benutzen Sie den Installer ausnahmsweise von der Shell aus. Dort wird die fehlerhafte Programmzeile ausgegeben, was allerdings nur manchmal hilft, da die beliebten Klammern meistens wesentlich später zu Fehlermeldungen führen. Lassen Sie sich davon aber nicht abschrecken, sondern machen Sie von der Möglichkeit Commodores Gebrauch, damit wir in Zukunft wirklich nur noch einen Software-Installateur benötigen und keine unterqualifizierten Schwarzarbeiter. ■

Im Griff: Kickstart-Umschaltplatinen

Gefahrlos umschalten

Endlich hat man eines der Update-Kits fürs neue Betriebssystem 2.0 errungen, schon tauchen die ersten Zweifel auf, ob der Schritt richtig war: Viele Programme bzw. Spiele verweigern ihren Dienst.

von Patrick Ohly

Um diesem Problem entgegenzuwirken, greift man oftmals auf Kickstart-Umschaltplatinen zurück. Im AMIGA-Magazin 3/92 finden Sie im übrigen eine Schaltung, die sich einfach nachbauen läßt. Ein Problem gibt's aber immer noch:

Nur bei ausgeschaltetem Computer ist die Umschaltung gefahrlos möglich. Ansonsten provoziert man einen schweren Systemabsturz mit unvorhersehbaren Folgen, da bei laufendem Computer ständig auf ROM-Routinen zugegriffen wird und diese nach dem Umschalten an anderen Stellen liegen. Das bedeutet also, Computer ausschalten, zehn Sekunden warten, wieder einschalten usw.

Abhilfe schafft das Programm »KickSwitch«. Es läßt sich vom CLI/Shell oder der Workbench starten. »KickSwitch« erwartet keine Argumente. Die auftauchende Sicherheitsabfrage ist entsprechend zu beantworten. Das Schließsymbol des Fensters veranlaßt KickSwitch zum Abbruch. Die Anwahl des OK-Schalters hingegen hält den Amiga an und der Schalter der Kickstart-Umschaltplatine ist nun gefahrlos umzulegen.

Der Amiga wird neu gestartet, nachdem man die linke Maustaste gedrückt hat. Dabei gehen allerdings alle Daten im Hauptspeicher verloren. Auch resetfeste Programme werden durch diesen Vorgang entfernt.

Das Programm läßt sich mit dem PD-C-Compiler Dice (Fish-Disk 491) übersetzen. Die drei erforderlichen Listings, »Kickswitch.h«, »Kickswitch.c« und »ColdReboot.asm« finden Sie auch auf unserer Diskette zum Heft (Seite 114). *rz*

Quell- und Literaturhinweise

[1] Commodore Amiga, AMIGA ROM Kernel Reference Manual Hardware, Third Edition, Reading 1991

[2] ColdReboot.asm (Fish-Disk 344)


```
; KickReboot.asm geschrieben von Patrick Ohly
; nach ColdReboot.asm von Commodore

ABSEXECBASE      EQU 4         ; Zeiger auf ExecBase
MAGIC_ROMEND     EQU $01000000  ; Ende Kickstart-ROM
MAGIC_SIZEOFFSET EQU -$14 ; Offset Ende des ROMs zur Größe
_LVOSupervisor   EQU -30  ; Offset der Supervisor-Funktion
_LVODisable      EQU -120 ; Offset der Disable-Funktion
CHKSUM           EQU $52  ; Offset für Library-Checksum
PRA              EQU $bfe001 ;Register für linkr Maustaste
LMB              EQU 6      ;Bit für linke Maustaste

   section text,code
   XDEF _KickReboot
_KickReboot:
   move.l  ABSEXECBASE,a6
   jsr _LVODisable(a6)   ; schalte Interrupts ab
   addq.w  #1,CHKSUM(a6) ; erzwingt Neuaufbau der ExecBase
   lea.l   Loop(pc),a5   ; Adresse des auszuführenden Codes
   jsr _LVOSupervisor(a6) ; Code anspringen
                          ; Funktion wird nicht beendet
   ds.l        0
Loop: move.b PRA,d0          ; Auf Mausklick warten ...
   btst #LMB,d0
   bne  Loop
   lea.l   MAGIC_ROMEND,a0  ; Adresse der Initialisierungs-
                            ; Routine im ROM ausrechnen ...
   sub.l   MAGIC_SIZEOFFSET(a0),a0
   move.l  4(a0),a0
   subq.l  #2,a0

   reset                     ; Reset ...

   jmp (a0)                  ; Routine anspringen
```


»KickReboot.asm«: Die Assemblerroutine führt einen Reset aus (alle Programme auch auf unserer PD-Diskette zu finden; s. S. 114)

```
/* KickSwitch V1.1
 * Autor: Patrick Ohly
 * Zu Kompilieren mit dem DICE-Compiler
 * Compileraufruf:
 * dcc KickSwitch.c KickReboot.asm -o KickSwitch
 */

#include <stdlib.h>
#include <exec/types.h>
#include <intuition/intuition.h>
#include <dos/dos.h>

#include <clib/exec_protos.h>
#include <clib/intuition_protos.h>
#include <clib/graphics_protos.h>

#include "KickSwitch.h"

VOID KickReboot(VOID);
struct Window *DialogWindow;

/* CloseAll schließt das Fenster und beendet Programm */
VOID CloseAll(VOID) {
  CloseWindow(DialogWindow);
  exit(RETURN_WARN);
}

int _main(VOID) {
  if(DialogWindow=OpenWindow(&DialogNewWindow)) {
     struct IntuiMessage *IMsg;
     WaitPort(DialogWindow->UserPort);
     while( IMsg= (struct IntuiMessage *)
             GetMsg(DialogWindow->UserPort)) {
        ULONG Class=IMsg->Class;
        APTR IAddress=IMsg->IAddress;
        ReplyMsg((struct Message *)IMsg);
     switch(Class) {
        case CLOSEWINDOW:
            CloseAll();
            break;
       case GADGETUP:
          if(IAddress==&OKGadget) {
            /* Schreibe neuen Text und löse Reset aus */
            ClearScreen(DialogWindow->RPort);
            PrintIText(DialogWindow->RPort, &InfoText1,0,0);
            KickReboot();
          } else CloseAll();
            break;
      default:
            break;
     }
   }
   }
  return(RETURN_FAIL);
}

/* keine spezielle Behandlung eines WB-Starts */
VOID _waitwbmsg(VOID);
VOID nevercalled(VOID) {
  _waitwbmsg();
}

int wbmain(VOID) {
  return(_main());
}
```


»Kickswitch.c«: Das Programm öffnet benötigte Ressourcen. U.a. importiert es die Datei »Kickswitch.h«.

```
/* Windowdefinition for KickSwitch.c */
struct TextAttr TOPAZ60 = {
  (STRPTR)"topaz.font",
  TOPAZ_SIXTY,0,0
};
/* 3D-Effekt */
/* daher zwei Teile für Rand */
SHORT GadgetVectorsLeftUpper[] = {
  82, 0, 0, 0, 0, 15
  };
 SHORT GadgetVectorsRightLower[] = {
  82, 1, 82, 15, 1, 15
  };
/* Nicht selektierter Rand: */
struct Border GadgetBorderNormal2 = {
  -1,-1, 1,0,JAM1,
  3, GadgetVectorsRightLower,
  NULL
};
struct Border GadgetBorderNormal1 = {
  -1,-1, 2,0,JAM1,
  3, GadgetVectorsLeftUpper,
  &GadgetBorderNormal2
};
/* Selektierter Rand: */
struct Border GadgetBorderSelected2 = {
  -1,-1, 2,0,JAM1,
  3, GadgetVectorsRightLower,
  NULL
};
struct Border GadgetBorderSelected1 = {
  -1,-1, 1,0,JAM1,
  3, GadgetVectorsLeftUpper,
  &GadgetBorderSelected2
};
/* Text für Cancel-Gadget: */
struct IntuiText CANCELText = {
  1,0,JAM2, 13,3,
  &TOPAZ60, (UBYTE *)"CANCEL",
  NULL
};

/* Cancel-Gadget: */
struct Gadget CANCELGadget = {
  NULL, 220,29, 83,14,
  GADGHIMAGE, RELVERIFY, BOOLGADGET,
  (APTR)&GadgetBorderNormal1,
  (APTR)&GadgetBorderSelected1,
  &CANCELText,
  0L, NULL, NULL, NULL
};
/* Text für OK-Gadget: */
struct IntuiText OKText = {
  1,0,JAM2, 29,3,
  &TOPAZ60, (UBYTE *)"OK",
  NULL
};
/* Sicherheitsabfrage: */
struct IntuiText QuestionText = {
  1,0,JAM2, -2,-19,
  &TOPAZ60, (UBYTE *)
      "Wollen Sie wirklich rebooten?",
  &OKText
};
/* OK-Gadget: */
struct Gadget OKGadget = {
  &CANCELGadget, 18,29, 81,14,
  GADGHIMAGE, RELVERIFY, BOOLGADGET,
  (APTR)&GadgetBorderNormal1,
  (APTR)&GadgetBorderSelected1,
  &QuestionText,
  0L, NULL, NULL, NULL
};
/* Hinweis-Text: */
struct IntuiText InfoText2 = {
  1,0,JAM2, 1,27,
  &TOPAZ60, (UBYTE *)
    "Drücken Sie die linke Maustaste!",
  NULL
};
struct IntuiText InfoText1 = {
  1,0,JAM2, 18,10,
  &TOPAZ60, (UBYTE *)
      "Wählen Sie ein Kickstart und",
  &InfoText2
};
/* Kommunikationsfenster: */
struct NewWindow DialogNewWindow = {
  0,0, 342,65, 0,1,
  GADGETUP+CLOSEWINDOW,
  WINDOWSIZING+WINDOWDRAG+WINDOWDEPTH+
  WINDOWCLOSE+GIMMEZEROZERO+ACTIVATE+
  NOCAREREFRESH,
  &OKGadget, NULL,
  (UBYTE *)
    "KickSwitch von V1.1 Patrick Ohly",
  NULL, NULL,
  5,5, -1,-1,
  WBENCHSCREEN
};
```


»Kickswitch.h«: Hier werden wichtige Definitionen fürs Programm vereinbart. Sie wird im Hauptprogramm eingebunden.

Programmierfehler vermeiden

Darauf sollten Sie achten

Bei der Programmierung des Amiga können sich sehr leicht Fehler einschleichen, die häufig lange unbemerkt bleiben, um dann plötzlich (z.B. bei neuen Kickstartversionen) um so schwerwiegender in Erscheinung zu treten. Wir nennen Ihnen einige Fehlerquellen, die Sie kennen (und vermeiden) sollten.

von Heinzjörg Rabe

Der Amiga bietet eine offene Systemarchitektur und kann in vielfältiger Weise erweitert oder modifiziert werden. Machen Sie keine Annahmen darüber, welche Speicherbereiche mit RAM bestückt sind. Im Zweifelsfalle kann die Funktion »TypeOfMem()« der Exec-Library verwendet werden, um zu erfahren, ob an einer gegebenen Adresse RAM existiert.

Jeder Zugriff auf unbelegte Speicherbereiche, also Adreßbereiche, die (momentan) weder mit ROM, noch RAM, noch irgendwelchen I/O-Chips (CIAs, Customchips, Echtzeituhr, etc.) belegt sind, kann folgenschwere Auswirkungen haben: Im günstigsten Falle passiert gar nichts, ein Schreibzugriff verändert kein einziges Byte, ein Lesezugriff liefert irgendwelche Zufallswerte.

Der Griff ins (Speicher-)Leere

Sehr viel häufiger bewirkt ein Schreibzugriff auf diese Bereiche jedoch, daß ein Byte an einer anderen Adresse ungewollt verändert wird. Von manchen »legalen« Adreßbereichen existieren mehrere sogenannte »Geisterbilder«, d.h. ein und dieselbe Speicherstelle läßt sich unter mehreren anderen Adressen ansprechen. Man kann sich allerdings nicht darauf verlassen, daß diese Geisterbilder bei allen Amigas und mit unterschiedlichen RAM-Ausbaustufen an derselben Adresse liegen (obwohl das leider einige kommerzielle (Spiel)-Programme zur Verwirrung möglicher Cracker gerne tun).

Alle Register der Customchips ($00DFF000-$00DFF1FF) können an der gleichen Adresse jeweils entweder nur gelesen (Read-Only) oder nur geschrieben (Write-Only) werden. Die meisten Register können nur beschrieben werden. Einige wenige können geschrieben und gelesen werden, allerdings gibt es dann zwei verschiedene Adressen: eine zum Lesen und eine zum Beschreiben des Registers. Genauergesagt besitzen die Customchips keine Read/Write-Signalleitung, um die Datenrichtung festzulegen. Die Datenrichtung wird einzig und allein durch die Adresse selbst bestimmt. Daraus erklärt sich dann auch folgendes Verhalten:

Ein Lesezugriff auf ein Nur-Schreibregister (Write-Only) wirkt wie das Schreiben eines Zufallswertes in dieses Register! Wenn Sie also den Wert in einem Customchip-Register verändern und sich vorher sicherheitshalber den alten Wert in dem Register merken wollen, denken Sie immer daran, daß Sie eine andere Adresse auslesen müssen, als Sie beschreiben wollen.

Häufig existiert kein entsprechendes Leseregister, so daß die aktuellen Registerinhalte nur aus den Strukturen entnommen werden können, durch die sie initialisiert wurden (z.B. die Farbregister). Es ist daher auch verboten, einzelne Bits von Customchip-Registern mit BSET oder BCLR bzw. unter Verwendung einer OR- bzw. AND-Maske zu ändern, da diese Befehle vorher einen Lesezugriff auf dieselbe Adresse versuchen (und dadurch einen Zufallswert in das Register schreiben)! Sie müßten dann schon das entsprechende Leseregister in ein Datenregister auslesen, das gewünschte Bit im Datenregister verändern und anschließend das geänderte Datenregister in das entsprechende Schreibregister der Customchips zurückschreiben.



Zur Vereinfachung dieser Problematik können bei bestimmten Registern (z.B. DMACON und INTENA) auch einzelne Bits durch eine spezielle Maske gesetzt bzw. gelöscht werden, ohne vorher die alten Werte auslesen zu müssen. Dabei bestimmt dann Bit 15 der Maske, ob gesetzt (1) oder gelöscht (0) werden soll, und jedes gesetzte Bit 0-14 der Maske kennzeichnet ein zu änderndes Bit.

Wenn Sie ein Customchip-Schreibregister mit dem Wert $0000 belegen wollen, beachten Sie bitte, daß der »CLR«-Befehl beim MC68000-Prozessor zuerst einen Lesezugriff ausführt (und dadurch einen Zufallswert in das Register schreibt) und erst danach den Wert $0000 schreibt. Verwenden Sie daher bitte

```
MOVE.W #$0000,...
```

und vergewissern Sie sich, daß Ihr Assembler dieses nicht zu

```
CLR.W ...
```

optimiert! Besonders bei den sog. Strobe-Registern kann das zu Problemen führen, da das Register hier ggf. zweimal getriggert wird. Da bei Strobe-Registern der geschriebene Wert egal ist, sollte man hier besser

```
MOVE.W #1,...
```

schreiben, was eine ungewollte Optimierung sicher verhindert.

Bitte seien Sie sich der Brisanz, die in den Customchips und den Geisterbildern steckt, bewußt: Ein einziger Lesebefehl auf eine unbelegte Adresse, an der sich zufällig ein Geisterbild eines Customchip-Registers befindet, kann den Computer zum Abstürzen bringen. Ein unbedachter Zugriff auf bestimmte Customchip-Register könnte z.B. die Interrupts sperren, so daß der Computer auf keine Tastatur- oder Mausbetätigungen mehr reagiert, oder den Blitter starten und dadurch große RAM-Bereiche überschreiben, oder den Bildschirmaufbau völlig durcheinander bringen (»Fireworks-Display«).

Seien Sie besonders vorsichtig, wenn Sie dem Benutzer in Ihrem Programm die Mög-

lichkeit geben, Adreßbereiche einzugeben, die dann durchsucht, angezeigt oder sonstwie bearbeitet werden sollen. Bei Disassembler-Monitoren kann ein einziger unbedachter Befehl (z.B. zu »großzügig« bemessene Bereichsgrenzen für Suchbefehle) das gesamte System lahmlegen. Besonders beachtenswert ist, daß auch reine Lesebefehle zum Absturz des Amiga führen können (Seka-Besitzer können ja mal »q $dff000« versuchen). Bei Intuition-orientierten Debuggern (z.B. »DBug« oder »Metascope«) führt z.B. auch ein unbedachtes Verschieben des Proportional-Gadgets in Memory-Windows zum Systemabsturz.

Flagzugriff: Bitte Bit für Bit

Häufig sind den einzelnen Bits von Speicherstellen (z.B. den Registern der CIAs) ganz bestimmte Aufgaben zugeordnet (z.B. Power-LED an/aus). Wenn Sie nur eine dieser Funktionen ändern möchten (z.B. Power-LED aus), dürfen Sie natürlich auch nur das eine, für die spezielle Aufgabe zuständige, Bit der Speicherstelle verändern; die anderen Bits müssen unverändert bleiben! Da man den Zustand dieser anderen Bits nicht vorhersehen kann (!), muß das gewünschte Bit entweder über BSET bzw. BCLR einzeln gesetzt bzw. gelöscht werden, oder es wird eine OR- bzw AND-Maske verwendet, um die anderen Bits unverändert zu lassen. Gehen Sie niemals davon aus, daß die anderen Bits bei jedem Lesen immer gleich bleiben (selbst wenn sie das zufällig mal sein sollten)!

Z.B. schaltet man so die Power-LED (und gleichzeitig auch den Audio-Filter) aus:

```
bset #1,$bfe001
```

Genausogut hätte man auch

```
ori.b #%00000010,$bfe001
```

verwenden können.

OR- und AND-Masken bieten den Vorteil, daß gleich mehrere Bits der Speicherstelle gleichzeitig verändert werden können.

Bitte beachten Sie dabei, daß der Amiga ein Multitasking-Betriebssystem hat. Wenn Sie z.B. den obigen Befehl in drei Befehle aufspalten würden

```
move.b $bfe001,d0
ori.b  #%00000010,d0
move.b d0,$bfe001
```

könnte es passieren, daß, nachdem Sie $00BFE001 ausgelesen haben, ein Taskwechsel erfolgt und ein anderer Task $00BFE001 ändert. Der vorher ausgelesene Wert in D0 wäre dann nicht mehr aktuell und das Zurückschreiben könnte dann einen schwerwiegenden Fehler bewirken.

Was kann man machen, um Fehler zu vermeiden? Nehmen Sie Änderungen von Speicherstellen, auf die möglicherweise auch ein anderer Task zugreifen könnte, daher möglichst in einem einzigen Befehl vor, oder sperren Sie (besonders bei Programmierung in Hochsprachen) notfalls vorher das Multitasking (Forbid). Man nennt das daher auch »in an atomic way« (zu deutsch etwa: auf untrennbare Art und Weise). Einige Felder spezieller Strukturen sind von Commodore ausdrücklich dazu freigegeben, »in an atomic way« geändert zu werden (z.B. darf man das »RMBTRAP«-Flag von Intuition-Windows so direkt manipulieren).

Prozessoren ab dem MC68010 besitzen ein sogenanntes Vector-Base-Register (VBR), das es erlaubt, die Exception-Vektoren, die beim MC68000-Prozessor immer im ersten KByte ($00000000-$000003ff) liegen, an jede andere Adresse zu verlegen. Ab OS 2.0 verlegt das Betriebssystem die Vector-Base normalerweise in das schnellste vorhandene RAM. Dadurch wird das Zeitverhalten, besonders bei der Annahme von Interrupts, verbessert, da der Prozessor beim Zugriff auf einen dieser Auto-Vektoren nicht auf die Freigabe des Chip-RAMs durch die Customchips zu warten braucht. Es war zwar schon immer unerlaubt, die Auto-Vektoren (Interrupt- Vektoren, Exception-Vektoren) direkt zu verändern, aber bei verändertem VBR führt das jetzt mit Sicherheit zu Fehlfunktionen.

Bis einschließlich Kickstart 1.3 konnte man ungefähr abschätzen, wo z.B. die Exec-Library oder der Supervisorstack angelegt werden, da hierfür nur einige festvorgegebene Adreßbereiche in Frage kamen. Seit OS 2.0 könnten diese Bereiche überall im RAM liegen.

Seit OS 2.0 ist das Betriebssystem wesentlich »intelligenter« und benutzerfreundlicher geworden. Windows werden auf Wunsch z.B. automatisch soweit verkleinert oder verschoben, bis sie auf den Screen passen. Das kann aber auch einige unerwünschte Effekte haben. Auch wenn ausdrücklich verlangt wird, z.B. ein Window auf eine bestimmte Größe zu bringen (SizeWindow), ist nicht sicher, daß es auch wirklich diese Größe hat. Z.B. läßt sich das Nicht-Backdropwindow der OS 2.0-Workbench auch durch direktes »SizeWindow()« nicht kleiner machen, als durch das Size-Gadget mit der Maus.

Routinen umlenken aber richtig

Viele Programme verwenden die Funktion »SetFunction()« der »exec.library«, um Sprungvektoren von Libraries auf eigenen Code umzuleiten, und die ursprüngliche Funktion dadurch zu patchen. Wenn Sie das auch machen wollen, beachten Sie bitte, daß es keine hundertprozentig sichere Methode gibt, einen solchen Patch wieder aufzuheben:

Stellen Sie sich z.B. vor, Ihr Programm würde die Funktion »OpenWindow()« patchen, um z.B. alle Windows automatisch auf einen eigenen Screen umzuleiten. Nehmen wir an (nur um der Einfachheit halber konkrete Werte zu haben), die alte Funktion OpenWindow() läge bei $00F85000 und ihre neue Funktion begänne bei $00081000. Ihr Programm würde dann (in diesem Beispiel) in der NewWindow-Struktur, die der gepatchten OpenWindow()-Funktion übergeben wurde, den Windowtyp auf CUSTOMSCREEN setzen, und die Adresse des eigenen Screens eintragen. Anschließend würde Ihr Programm die originale OpenWindow()-Funktion anspringen, also JMP $00F85000.

Nehmen wir weiter an, jetzt würde ein anderes Programm gestartet, das ebenfalls die OpenWindow()-Funktion patched, um z.B. jedes neugeöffnete Window mit einem Size-Gadget zu versehen. Dieses Programm (nennen wir es »Programm B«) erhält von SetFunction() als alte Adresse der original OpenWindow()-Funktion jetzt natürlich nicht $00F85000 geliefert, sondern (wir hatten die Funktion ja bereits gepatched) $00081000. Anschließend würde dann vielleicht noch »Programm C« gestartet, das ebenfalls dieselbe Funktion patched. Wenn nun die OpenWindow()-Funktion aufgerufen wird, wird zuerst der Patch von Programm C ausgeführt, dann der von Programm B, dann der Ihres Programms, und schließlich wird die original OpenWindow()-Funktion ausgeführt.

Stellen Sie sich nun vor, was passieren würde, wenn Sie »Programm B« beenden? Zuerst stellt das Programm den OpenWindow-Vektor wieder auf den Wert zurück, den es beim Aufruf von SetFunction dort vorgefunden hat, also zeigt er jetzt in Ihr Programm. Wenn Programm B anschließend terminiert, wird der Speicher, den das Programm beansprucht hatte, wieder freigegeben und könnte z.B. von anderen Tasks alloziert und überschrieben werden. Die OpenWindow()-Funktion würde jetzt nie mehr den Code von Programm C ausführen, da dieses ja aus der »Umleitungskette« entfernt wurde. Wenn nun aber Programm C enden soll, so würde es auch den OpenWindow-Vektor wieder auf den alten Wert zurücksetzen, den SetFunction() geliefert hatte. Dieser zeigt aber auf Programm B, das ja bereits beendet, und dessen ursprünglicher Code im Speicher natürlich überschrieben wurde. Beim nächsten Aufruf der OpenWindow()- Funktion würde dieser sinnlose Code angesprungen, und ein böser Systemabsturz wäre die Folge.

Eine relativ sichere Lösung für dieses Problem, ist folgende: Wenn Ihr Programm beendet werden soll, prüfen Sie, ob der von Ihrem Programm gepatchte Vektor noch immer auf die Adresse zeigt, die Sie bei SetFunction angegeben haben (im eigenen Programm). Wenn ja, können Sie den Vektor wieder auf den Original-Wert setzen und Ihr Programm gefahrlos beenden, denn mit Sicherheit hat nach Ihnen kein weiteres Programm denselben Vektor gepatched (oder aber es ist bereits wieder beendet worden). Zeigt der Vektor auf eine andere Adresse, so hat ein anderes Programm denselben Vektor nach Ihnen gepatched. Wenn Sie jetzt einfach Ihr Programm beenden würden, würde der nächste OpenWindow()-Aufruf ins Leere springen und mit Sicherheit zum Guru führen. Daher informieren Sie einfach den Benutzer, daß ein Beenden momentan leider (noch) nicht möglich ist, und empfehlen Sie ihm, doch bitte zuerst ein anderes Programm zu beenden, daß möglicherweise den Vektor später gepatched haben könnte. Anschließend können Sie entweder wieder normal mit Ihrem

Programm fortfahren, oder in regelmäßigen Abständen (durch Delay()-Verzögerungen) den Vektor prüfen, bis er den gewünschten Wert hat. Eine gewisse Gefahr besteht trotzdem noch, falls ein anderes Programm noch Ihren Code (die gepatchte Funktion) ausführt, während Ihr Programm endet. Daher sollte man nach Zurücksetzen des gepatchten Vektors auf den alten Wert, noch etwas warten (z.B. der Aufruf Delay(500) ist mehr als ausreichend), bevor man das Programm beendet und dadurch den Speicherbereich zum Überschreiben durch andere Tasks freigibt.

Um sich davor zu schützen, selbst mehrfach gestartet zu werden, was natürlich meistens nicht besonders sinnvoll wäre, sollte Ihr Programm am Anfang prüfen, ob es bereits läuft. Dazu wird in der Regel ein öffentlicher Messageport mit einem eindeutigen Namen eingerichtet, nach dem das Programm jedesmal sucht, und abbricht, wenn er bereits existiert.

Watschen für´s Patchen

Um eins ganz klar festzustellen: Es ist kein gutes Zeichen großer Programmierkunst, Systemfunktionen zu patchen, nur um ein spezielles Programm, das die gepatchte Funktion irgendwann einmal benutzt, etwas zu verbessern. Hier sei an die zahllosen Programme erinnert, die AmigaBASIC etwas benutzerfreundlicher machen wollten (eigener Filerequester, größerer Cursor, etc.) und dafür jeweils wichtige Systemfunktionen (OpenWindow(), RectFill()) gepatched haben. Viel besser wäre es, das betreffende Programm selbst zu patchen.

Fast alle Funktionen der Libraries liefern ein Ergebnis zurück, das dann immer im Prozessorregister D0 übergeben wird. Häufig bedeutet dabei der Wert Null, daß ein Fehler aufgetreten ist. Bitte beachten Sie als Assemblerprogrammierer, daß die Statusflags (insbesondere das Zeroflag) des Prozessors dabei in der Regel nicht entsprechend dem Wert in D0 gesetzt sind. Sie müssen D0 daher selbst auf Null überprüfen, z.B. durch TST.L. Beachten Sie bitte auch, daß der MOVE-Befehl das Zeroflag nur dann aktualisiert, wenn ein Transport von Daten stattgefunden hat.

```
move.l d0,DOSBase
beq.s  Fehler_noDOS
```

ist also korrekt,

```
move.l d0,a6
beq.s  Fehler
```

aber nicht, weil der MOVE-Befehl hier nicht das Zeroflag aktualisiert.

Wenn ein Adreßzeiger (APTR) in einer Struktur den Wert Null enthält, bedeutet das, daß dieses Feld der Struktur unbenutzt ist. Prüfen Sie diesen Sonderfall immer, bevor Sie evtl. die Adresse Null referenzieren. Dieser Fehler wird z.B. häufig gemacht, wenn beim Start eines Programmes von der Workbench der Name eines ggf. durch die erweiterte Auswahl selektierten Icons ermittelt werden soll.

Die Speicherstelle Null hat im Betriebssystem des Amiga lediglich eine Funktion, nämlich im Falle des automatischen Neustarts durch einen fatalen Fehler (Guru) zu bewirken, daß vorher die entsprechende blinkende Alert-Meldung (früher auch »Guru-Meditation« genannt) angezeigt wird. Dazu wird vor dem Auslösen des ColdReboot das Wörtchen »HELP« in die Speicherstellen $00000000-$00000003 geschrieben. Findet die Reset-Routine später dieses Wort, so wird erst die Alert-Meldung angezeigt, bevor der Bootvorgang fortgesetzt wird. In jedem Fall wird das Longword ab $00000000 wieder mit Null gelöscht, bevor der Benutzer die Kontrolle über den Amiga zurückerhält. (Normalerweise holt ein MC680x0-Prozessor beim Reset aus dem ersten Longword des Adreßbereichs seinen initialen Supervisor-Stackpointer. Durch das Memory-Overlay des Amiga wird beim Reset aber bekanntlich dort kurzzeitig das ROM eingeblendet.)

Verwenden Sie die einzelnen Felder von Daten-Strukturen des Betriebssystems nur, wenn diese ausdrücklich zur Verwendung freigegeben sind, und nur unter den jeweils dokumentierten Bedingungen. Besondere Vorsicht ist geboten, wenn in den Feldern Adreßzeiger erwartet werden, und man über diese vermeintlichen Zeiger auf andere Daten referenziert. Als Beispiel sei hier ein sehr verbreiteter Fehler angeführt, der zum Versagen vieler Kickstart 1.3-Programme unter OS 2.0 und höher führt:

Die Programme haben für ihr Window u.a. folgende IDCMP-Flags gesetzt:
- IDCMP_GADGETDOWN,
- IDCMP_GADGETUP,
- IDCMP_MOUSEBUTTONS.

Wenn der Benutzer nun eine mit diesen Flags assoziierte Funktion ausführt, also z.B. ein Gadget anklickt, oder eine Maustaste drückt, sendet Intuition eine Message an den Userport des Windows. Das Programm holt diese Message mit »GetMsg()« vom Messageport ab, kopiert relevante Teile der Message in eigene Speicherbereiche und sendet die Message dann mit »ReplyMsg()« wieder zurück.

Falls die Message durch ein angeklicktes Gadget ausgelöst wurde (IDCMP_GADGETUP oder IDCMP_GADGETDOWN), enthält das Feld »im_IAddress« einen Pointer auf die Gadget-Struktur. Das Feld »gg_GadgetID« der Gadget-Struktur enthält eine vom Programmierer festgelegte Nummer, anhand derer das Programm das betreffende Gadget identifizieren kann.

Da Intuition den durch eine Intuition-Message belegten Speicher erst für eine neue Message verwenden kann, wenn der Empfänger die Message mit »ReplyMsg()« beantwortet hat, sollte das Programm die Message so schnell wie möglich beantworten (nachdem vorher die benötigten Daten aus der Message kopiert wurden). Die fehlerhaften Programme wollen hier etwas »zu schnell« sein, denn sie verzichten, um die vermeintliche GadgetID zu erhalten, auf einen Test, ob die Message überhaupt von einem Gadget ausgelöst wurde.

Doch nun kommt's: Wenn die Intuition-Message nicht durch ein angeklicktes Gadget ausgelöst wurde, sondern z.B. durch einen Mausklick an eine freie Stelle des Windows (IDCMP_MOUSEBUTTONS), ist das Feld »im_IAddress« ungültig! Offiziell gilt »im_IAddress« in diesem Falle schon immer als undefiniert, jedoch haben sich manche Programmierer auf ein ganz bestimmtes undokumentiertes Verhalten verlassen.

Bis einschließlich Kickstart 1.3 enthielt »im_IAddress« im Fall eines IDCMP_-MOUSEBUTTONS-Events den Wert Null. Das fehlerhafte Programm hat dann also die vermeintliche GadgetID nicht aus einer Gadget-Struktur gelesen, sondern aus der absoluten Adresse $00000026 (NULL + gg_GadgetID). Bei Prozessoren ohne VBR, also dem MC68000, oder bei VBR=0, befinden sich an dieser Adresse die Bits 0-15 des Trace-Exception-Vektors. (Vektor 9 = VBR+$024)

Bis Kickstart 1.3 lesen Programme mit diesem Programmierfehler also »lediglich« eine falsche GadgetID. Da es unwahrscheinlich ist, daß ein Programm zufällig ein Gadget mit dieser ID ($0826 bei Kickstart 1.3) verwendet, und die Programme später doch noch die »im_Class« der Message überprüfen, reagieren die Programme trotzdem wie beabsichtigt.

Macken mit der Maus

Ab OS 2.0 ist »im_IAddress« aber im Falle eines IDCMP_MOUSEBUTTONS-Events nicht mehr Null. Unter OS 2.04 (V37.175; es ist nicht sicher, daß das unter späteren Kickstart-Versionen so bleibt!), enthält »im_IAddress« dann »Delta-Mouse«-Koordinaten: Die Strecke, um die die Maus vom Moment des Drückens bzw. Loslassens der Maustaste, bis zum Versenden der entsprechenden Intuition-Message, bewegt wurde, wird in »im_IAddress« abgelegt. Der X-Anteil im höherwertigeren Teil (Bits 16-31) und der Y-Anteil im niederwertigeren Teil (Bits 0-15). Dabei wird eine Bewegung nach links oben negativ, und nach rechts unten positiv eingetragen. Jeder ungerade Wert könnte dann einen Address-Error bewirken.

Ein Beispiel: Nehmen wir an, Sie bewegen die Maus relativ langsam nach unten und drücken dann an einer Stelle des Windows, die kein Gadget enthält, die linke Maustaste. In der Intuition-Message (vom Typ »IDCMP_MOUSEBUTTONS«), die das fehlerhafte Programm dann erhalten würde, enthält das Feld »im_IAddress« (Offset $1C) den Wert $00000001. Ein Befehl wie

```
MOVEA.L $1C(A2),A0
```

liest dann diesen vermeintlichen Pointer auf eine Gadget-Struktur aus, das Adreßregister A0 enthält also $00000001. Ein nachfolgender Befehl wie z.B.

```
MOVE.W  $26(A0),(A3)
```

soll eigentlich das Feld »gg_GadgetID« (Offset $26) der Gadget-Struktur auslesen (und in das Word schreiben, auf das A3 zeigt). Da A0 aber $00000001 ist, versucht der Prozessor nun, ein Word an der Adresse $00000027 ($0026+$00000001) zu lesen. Abgesehen da-

von, daß sich an dieser Adresse niemals eine Gadget-Struktur befindet, ja die Adresse selbst sogar ungültig sein könnte (z.B nichtbestückter RAM-Bereich), ist die Adresse ungerade!!! Ein MC68000-Prozessor löst also eine Address-Error-Exception aus, die normalerweise zum gefürchteten »Software Failure«-Requester mit Error #80000003 führt.

Um diesen Fehler zu vermeiden, muß vor dem Referenzieren des vermeintlichen Pointers auf die Gadget-Struktur geprüft werden, ob die Message vom Typ (im_Class) IDCMP_GADGETUP oder IDCMP_GADGETDOWN ist, und nur dann darf der Versuch unternommen werden, das Feld »im_IAddress« als Pointer auf eine Gadget-Struktur zu interpretieren, um dort die »gg_GadgetID« auszulesen.

Wenn Sie selbst ein vollausgebautes System (Festplatte, mehrere Diskettenlaufwerke, etliche Megabyte RAM, etc.) besitzen, bedenken Sie bitte auch, daß nicht jeder Amiga-Besitzer in dieser glücklichen Lage ist. Besitzer von nur einem Diskettenlaufwerk haben häufig bestimmte Files, die sie seltener benötigen, von Ihrer Boot-Diskette (normalerweise eine Kopie der Workbench-Diskette) gelöscht, um Platz für wichtigere Files zu erhalten. Geben Sie in der Dokumentation Ihrer Programme daher möglichst an, wenn das Programm selten gebrauchte Files nachladen möchte. Ein sehr bekanntes Programm startet z.B. CLI-Befehle durch die Execute()-Funktion der DOS-Library. Da diese Funktion keine Pfade (Path-Befehl) berücksichtigt, startet das Programm vorher den CLI-Befehl »Which«, um den expliziten Pfad zu erhalten, und hängt sich sang- und klanglos auf, wenn es den »Which«-Befehl nicht findet. Bedenken Sie, daß für Besitzer von nur einem Diskettenlaufwerk nahezu jedes nachzuladende File mit einem Diskettenwechsel verbunden und daher sehr nervtötend ist. Leider finden sich häufig Programme, die Ressourcen anfordern, die sie nicht benötigen, z.B. braucht ein Programm, das vom CLI/Shell gestartet wurde, in der Regel nicht die Icon-Library, da sie meist nur zum Nachladen des ».info«-Files und Auswerten der Tool-Types beim Start von der Workbench benötigt wird.

Wenn Ihr Programm von der Workbench gestartet werden kann und Tool-Types aus dem ».info«-File auswertet, aber sonst keine weiteren Files mehr nachzuladen braucht, sollten Sie das aktuelle Directory mit CurrentDir() bereits nach dem Laden des ».info«-Files (GetDiskObject) wieder auf den initialen Wert zurückstellen und nicht erst am Ende des Programms. Sonst muß der Benutzer beim Beenden Ihres Programms extra nocheinmal die Diskette mit Ihrem Programm einlegen. Das ist hochgradig frustrierend, zumal ja überhaupt nichts nachgeladen zu werden braucht. (Lediglich die Diskette muß eingelegt sein, um das Current Directory zuweisen zu können.)

Konfigurationsfiles für spezielle Voreinstellungen sind eine feine Sache, aber es wäre wünschenswert, wenn Ihr Programm diese Files nicht nur in »S:« oder »Devs:«, sondern auch im aktuellen Directory suchen würde. Besitzer nur eines Diskettenlaufwerks ersparen sich (wenn Sie Ihr Programm nicht auf eine Workbench-Diskette kopieren wollen), jedesmal einen Diskettenwechsel. Außerdem braucht man dann zum ersten Ausprobieren des Programms keine umständlichen Installationen (Konfigurationsfile kopieren) durchzuführen.

Wechseln Sie nie eine Diskette, während das Betriebssystem gerade lesend oder schreibend auf sie zugreift. Bei Programmen, die das Trackdisk-Device umgehen und die Diskettenlaufwerke direkt ansprechen, kann die Drive-LED (Leuchtdiode) leider oft nur nur als grober Hinweis darauf verstanden werden, ob momentan auf die eingelegte Diskette zugegriffen wird, oder nicht (die Diskette also gewechselt werden darf). Die LED des eingebauten Laufwerks (»DF0«) läßt sich völlig unabhängig vom Antriebsmotor ein- und ausschalten. Am Anschluß für die externen Diskettenlaufwerke »DF1«-»DF3« existiert keine Leitung zum Anschluß einer LED. Daher wird die LED häufig parallel zum Laufwerksmotor geschaltet, d.h. die LED leuchtet, wenn der Motor läuft. Manche Fremdhersteller steuern die LED aber auch durch die Drive-Select-Leitung an, wobei die Leuchtdauer häufig noch durch ein Monoflop auf etwa 1/2 Sekunde verlängert wird. Wenn ein Programm nun ca. 2-mal pro Sekunde das Laufwerk anspricht (z.B. um den Laufwerksmotor explizit auszuschalten), kann die LED auch ständig leuchten, obwohl der Laufwerksmotor aus ist, und nicht auf die Diskette zugegriffen wird.

Aktivieren Sie bei wichtigen Disketten möglichst immer den Schreibschutz (durch Verschieben des kleinen Plastikschiebers in der oberen Ecke der Diskette, so daß das rechteckige Loch frei ist). Entgegen Gerüchten aus der Virus-Szene ist es unmöglich (auch unter Umgehung des Betriebssystems) auf eine so schreibgeschützte Diskette zu schreiben, bzw. Daten zu löschen und dadurch zu zerstören.

Schreibschutz bei Spielen

Vor allem bei neugekaufter Original-Software (besonders Spielen) sollten Sie als erstes prüfen, ob der Schreibschutzschieber auch in Schreibschutzstellung ist. Unverständlicherweise ist das leider meistens nicht der Fall, die Hersteller wollen wohl die zwei Sekunden Arbeitszeit pro Diskette einsparen. Bei einigen Disketten (z.B. den neueren Workbench-Disketten) fehlt der Schreibschutzschieber ganz, so daß diese Disketten mit einem normalen (unmanipulierten) Diskettenlaufwerk überhaupt nicht beschrieben werden können.

Manche Programme, speziell Grafik-Demos, sperren während ihrer Laufzeit das Multitasking (Forbid), um einen fließenderen Bewegungsablauf der Animationen zu erzielen. Leider wird dabei häufig vergessen, zu warten, bis das Diskettenlaufwerk, von dem das Programm gerade geladen wurde, zum Stillstand gekommen ist. Der Laufwerksmotor dreht sich dann solange, bis das Programm endet und das Multitasking wieder eingeschaltet wird.

Anders als bei Festplatten, bei denen die Schreib-/Leseköpfe auf einem Luftkissen über den Platten schweben und diese dadurch in keinster Weise mechanisch strapazieren, wird der Schreib-/Lesekopf bei den Diskettenlaufwerken durch eine Feder gegen die Oberflächen der Diskette gedrückt, was auf die Dauer natürlich zur Abnutzung der magnetischen Beschichtung der Disketten und Verschmutzung bzw. Abrieb des Schreib-/Lesekopfes führt. Sollten auch Sie einmal das Multitasking, oder gar die Interrupts, für längere Zeit sperren wollen (müssen), schalten Sie anschließend explizit die Laufwerksmotoren aus. Auch vor einem selbst absichtlich per Software ausgelösten Reset [4] sollten Sie (nach Sperren der Interrupts) die Laufwerksmotoren ausschalten.

Hier ein Assemblerprogramm zum Ausschalten aller Diskettenlaufwerksmotoren (bei eingeschaltetem Multitasking ist es wirkungslos und sollte dann nicht angewendet werden):

```
moveq #3,d0       ;Bitnummer
DriveLoop:
 bset #7,$bfd100 ;DiskMotor aus
 bclr d0,$bfd100 ;kurzer Impuls
 bset d0,$bfd100 ; auf DriveSelect
 addq.b #1,d0
 cmpi.b #7,d0
 bne.s DriveLoop ;nächstes Laufwerk
```

Beachten Sie, daß es nur eine gemeinsame Signalleitung für alle vier Laufwerksmotoren gibt (Bit 7 von $BFD100). Daher ist jedes Diskettenlaufwerk mit einem Flipflop ausgerüstet, das den Zustand der Motor-Signalleitung in dem Moment speichert, wo das jeweilige Laufwerk selektiert wird. Deshalb sind jeweils drei Schreibzugriffe auf $BFD100 nötig, um einen Laufwerksmotor (sicher) auszuschalten.

Seit OS 2.0 läuft der Laufwerksmotor der Diskettenlaufwerke nach Beendigung des letzten Schreib-/Lesezugriffs auf die Diskette, nicht mehr so lange nach, wie vorher. Das hat Vor- und Nachteile. Wenn Files während dem Laden gleich bearbeitet (z.B. Programmfiles entpackt, IFF-Bilddaten aufbereitet, oder ASCII-Files für einen Texteditor umformatiert) werden, dauert die Bearbeitung auf 68000-Rechnern häufig so lange, daß der Floppymotor ab OS 2.0 immer kurzzeitig stehenbleibt und dann wieder anläuft. Dadurch dauert der Ladevorgang oft erheblich länger, als unter Kickstart 1.3. Programmierer sollten diesem Verhalten dadurch Rechnung tragen, daß sie vor dem Laden genug Speicher allozieren (aber möglichst nicht in einem Stück), die Datei komplett in den allozierten Speicherbereich laden, und erst dann bearbeiten, oder ihre Routinen soweit beschleunigen, daß der jeweils geladene Teilbereich in der Nachlaufzeit des Floppymotors (maximal 1 Sekunde) vollständig bearbeitet werden kann. *ub*

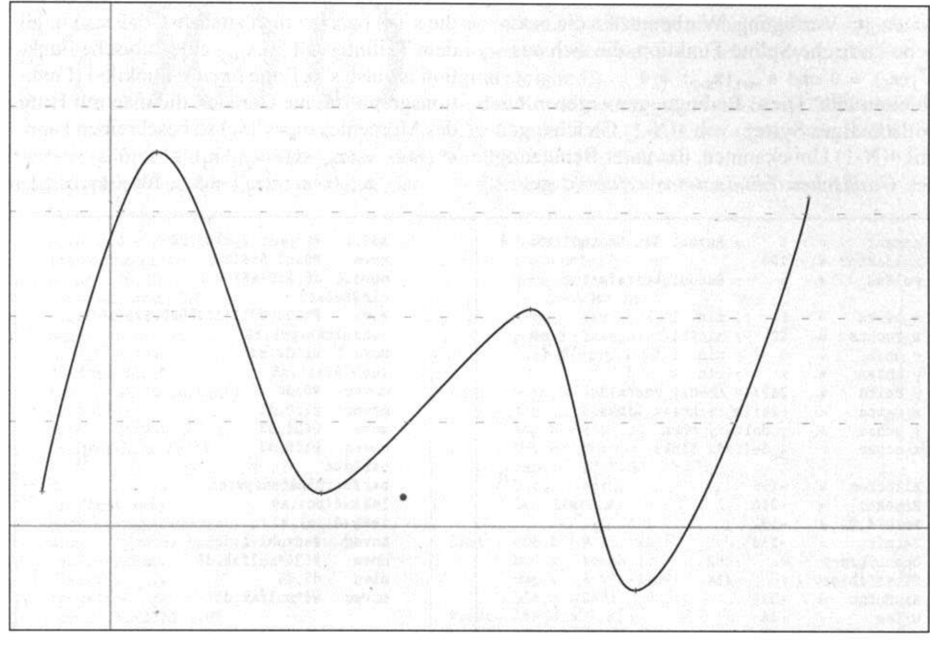
Literaturhinweise:
[1] Commodore-Amiga Inc.: AMIGA ROM Kernel Reference Manual: Libraries, Third Edition, Addison-Wesley, ISBN 0-201-56774-1
[2] Babel, Ralph: Das Amiga-Guru-Buch, Selbstverlag, Taunusstein
[3] Babel, Ralph: Kompatibilitätsrisiken, AMIGA-Magazin 5,6,7,10/90, Markt & Technik Verlag AG, München
[4] Zeitler, Rainer: Systemkonforme Programmierung, Faszination Programmieren Nr.1 (AMIGA-Sonderheft 1/93), Seite 87 f., Markt & Technik Verlag AG, München
[5] Zeitler, Rainer: Die goldenen Regeln, AMIGA-Magazin 2/93 Seite 44 ff., Markt & Technik Verlag AG, München

Mathematische Lösungen: Splines

Amiga kratzt die Kurve

Wie berechnet man Kurven mit dem Computer? Dazu gehört sicher eine Menge Mathematik. Der folgende Artikel erläutert die Grundlagen und zeigt, wie man das Ganze in Assembler umsetzt, wobei wir fleißig von Copper und Blitter zur Bildschirmdarstellung Gebrauch machen.

von Sebastian Wedeniwski



»Splines«: Ein Programm, das Spline-Funktionen grafisch darstellt ...

Das englische Wort »spline« bezeichnet eine mechanische Vorrichtung, die von Zeichnern benutzt wird, um glatte Kurven durch endlich viele Punkte (Knoten) zu zeichnen. Kurz: Eine Art elastisches Kurvenlineal. Hingegen ist die Spline-Interpolation das mathematische Gegenstück zu diesem Prozeß und verbindet gegebene Punkte durch eine »möglichst glatte« Kurve (kleines Bild rechts).

Als erstes wollen wir die allgemeine Aufgabe der Kurvenanpassung betrachten. Prinzipiell ist die Bestimmung einer Funktion zu beschreiben, die in einer Menge von gegebenen Punkten eine Menge von gegebenen Werte annimmt, d.h. wenn die Punkte $x_1,x_2,...,x_n$ und die zugehörigen Werte $y_1,y_2,...,y_n$ gegeben sind, ist eine Funktion zu finden, so daß gilt

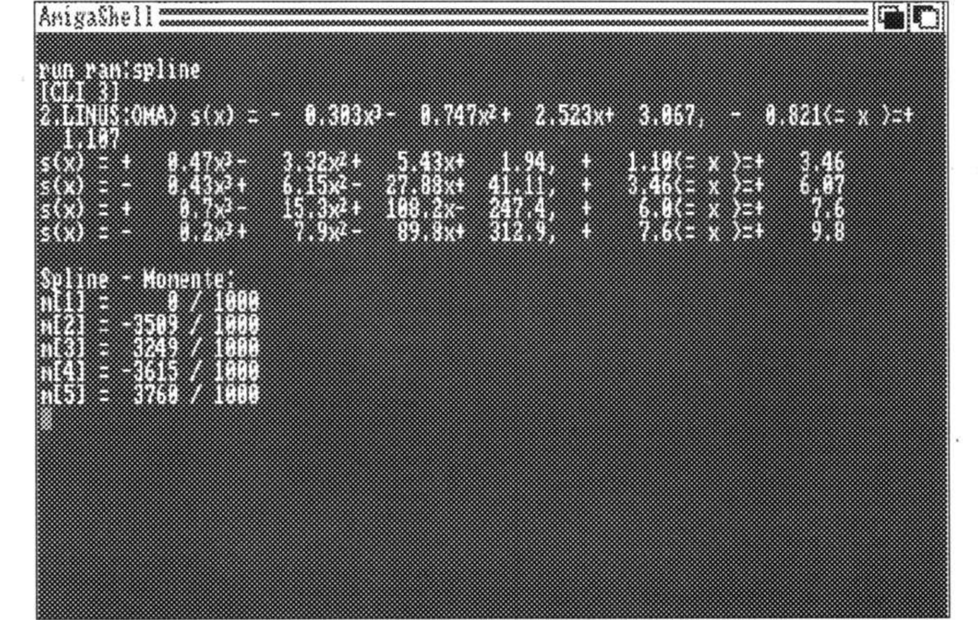
$$f(x_1)=y_1,\ f(x_2)=y_2,...,f(x_n)=y_n$$

In der Anwendung wird dann eine Tabelle von exakten Werten gespeichert, und andere Werte werden mittels einer Funktion bestimmt.

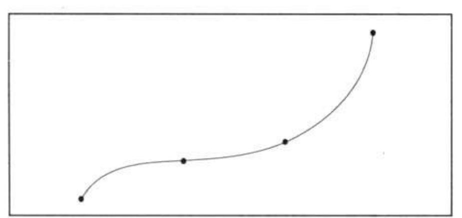
Die interpolierende Funktion von f(x) im Intervall [a,b] wird unter der sog. Spline-Funktion s(x) gesucht, wobei für das System der Knoten gilt:

$$a = x_1 < x_2 < ... < x_N = b$$

Betrachten wir nun die angenommene Form zwischen zwei benachbarten Knoten, so entspricht sie einem Polynom dritten Grades – d. h. einem kubischen Polynom. Infolgedessen ist eine Spline-Funktion stückweise aus n kubischen Polynomen so zusammengesetzt, daß die Funktion s(x) selbst und ihre beiden ersten Ableitungen (Steigungen des Funktionsgraphen) an den Stellen x_i (i=2,...,N-1) keine Sprungstellen besitzen. Mit anderen Worten ist die Funktion s(x) zweimal stetig differenzierbar (Eine stetige Funktion kann man sich »ohne abzusetzen« gezeichnet vorstellen). Neben



... und die Kurvenparameter berechnet und auf dem Bildschirm ausgibt



Beispiel: Kurve auf dem kürzesten Weg

kubischen Spline-Funktionen werden in der Literatur auch höhere Spline-Funktionen betrachtet. Der Einfachheit halber beschränken wir uns auf den kubischen Fall. Jedoch lassen sich die meisten Resultate leicht auf den allgemeinen Fall – Polynom höheren Grades – übertragen. In den Anwendungen genügt es häufig, die kubische Spline-Funktion anzuwenden, die sich aus N-1 verschiedenen kubischen Polynomen

$$s_j(x) = a_jx^3+b_jx^2+c_jx+d_j\ ,\ j = 1,2,..,N-1$$

zusammensetzt, wobei $s_j(x)$ als das kubische Polynom definiert ist, welches im Teilintervall $[x_j,x_{j-1}]$ zu verwenden ist. Also besteht die Er-

zeugung eines Splines in der Berechnung der Koeffizienten a, b, c, d aus den gegebenen Punkten x und Werten y, z.B. muß offenbar

$$s_j(x_j) = y_j \text{ und } s_j(x_{j+1}) = y_{j+1}$$

gelten, da die Kurve die Knoten berühren muß. Hinzu muß die zweite Ableitung der Polynome – oft als Moment bezeichnet – in den Knoten übereinstimmen. Um die Randbedingungen zu beschreiben, stehen verschiedene Möglichkeiten zur Verfügung. Wir benutzten die natürliche kubische Spline-Funktion, die sich aus

$$s''_1(x_1) = 0 \text{ und } s''_{N-1}(x_{N-1}) = 0$$

ableiten läßt. Diese Bedingungen ergeben ein vollständiges System von 4(N-1) Gleichungen mit 4(N-1) Unbekannten, das unter Benutzung des *Gaußschen Eliminationsverfahren* gelöst werden könnte, um alle Koeffizienten zu berechnen, die die Spline-Funktion beschreiben.

Dieselbe Spline-Funktion kann jedoch berechnet werden, da man mit Hilfe von

$$s''_k(x_k) \quad (k = 1,2,\ldots,N)$$

allein die interessierende Spline-Funktion s(x) leicht angeben kann, und damit sind in Wirklichkeit nur N-2 Unbekannte vorhanden. Zur Herleitung von Bestimmungsgleichungen für die s"(x) beachte man zunächst, daß s(x) in jedem Teilintervall $[x_j,x_{j+1}]$ eine kubische Funktion ist, also s"(x) eine *lineare* Funktion (Funktionsgraph ist eine Gerade), die man mit Hilfe des Momentes $m_k=s''(x_k)$ so beschreiben kann:

$$s''(x) = m_j(x_{j+1}-x)/(x_{j+1}-x_j) + m_{j+1}(x-x_j)/(x_{j+1}-x_j) \; ; \text{ für } x \varepsilon [x_j,x_{j+1}]$$

Durch Integration erhält man

$$s'(x) = -m_j(x_{j+1}-x)^2/2(x_{j+1}-x_j) + m_j+1(x-x_j)^2/2(x_{j+1}-x_j) + A_j$$

$$s(x) = m_j(x_j+1-x)^3/6(x_{j+1}-x_j) + mj+1(x-xj)^3/6(xj+1-xj) + A_j(x-x_j) + B_j$$

mit den Integrationskonstanten A_j und B_j.

Da $s(x_j) = y_j$ und $s(x_{j+1}) = y_{j+1}$ gilt, erhält man für A_j und B_j die Gleichungen:

$$B_j = y_j - m_j\,(x_{j+1}-x_j)^2\,/6$$

$$A_j = (y_{j+1}-y_j)/(x_{j+1}-x_j)-(x_{j+1}-x_j)(m_{j+1}-m_j)/6$$

Durch Einsetzen der Integrationskonstanten, Substituieren von $w = (x-x_j)/(x_{j+1}-x_j)$ und weitere mathematische Vereinfachungen, kann man die Spline-Funktion anschließend wie folgt formulieren:

```
anzahl      =   6   ; Anzahl der Knoten (min. 4)
mulfaktor   =   100
mulfak      =   5   ; Genauigkeitsfaktor

x_links     =   1   ; min. 1
x_rechts    =   10  ; min. 1
y_oben      =   5   ; min. 1
y_unten     =   1   ; min. 1
y_delta     =   240/(y_oben+y_unten)
x_delta     =   620/(x_rechts+x_links)
y_achse     =   y_delta*y_oben
x_achse     =   x_delta*x_links

AllocMem    =   -198
FreeMem     =   -210
Forbid      =   -132
Permit      =   -138
OpenLibrary =       -552
CloseLibrary =      -414
RawDoFmt    =   -522
Write       =   -48
Output      =   -60

breite      =   88
bs          =   breite*312

        move.l  4.w,a6
        move.l  #Copper_Ende-sprite+5*bs,d0
        move.l  #$10002,d1
        jsr AllocMem(a6)
        move.l  d0,copperBase
        beq Ende
        add.l   #Copper_Ende-sprite,d0
        move.l  d0,bild
        add.l   #2*bs,d0
        move.l  d0,bild2
        lea Copper(pc),a1
        move    d0,6(a1)
        swap    d0
        move    d0,2(a1)
        swap    d0
        add.l   #bs,d0
        lea punkte(pc),a0
        moveq   #anzahl-1,d7

Int_pk: move.l  d0,(a0)
        addq.l  #6,a0
        dbra    d7,Int_pk

        move    d0,14(a1)
        swap    d0
        move    d0,10(a1)
        swap    d0
        add.l   #bs,d0
        move    d0,22(a1)
        swap    d0
        move    d0,18(a1)
        move.l  copperBase,d0
        move    d0,30(a1)
        swap    d0
        move    d0,26(a1)
        swap    d0
        lea sprite(pc),a1
        move.l  d0,a0
        move    #Copper_Ende-sprite-1,d7

Copy:   move.b  (a1)+,(a0)+
        dbra    d7,Copy
        jsr Forbid(a6)
        lea $dff000,a6
        bsr Init
        move.l  copperBase,d0

        add.l   #Copper-sprite,d0
        move    #$3e0,$96(a6)
        move.l  d0,$80(a6)
        clr $88(a6)
        move    #%1000001111100000,$96(a6)
        lea mulu88(pc),a0
        move.l  bild2,a4
        lea 978(a4),a5
        moveq   #0,d0
        moveq   #100,d1
        move    #624,d2
        moveq   #100,d3
        bsr Line
        bsr Koordinatensystem
        lea x+4(pc),a0
        lea y+4(pc),a1
        moveq   #anzahl-1,d7
        move    #624*mulfak,d6
        divu    d7,d6
        moveq   #4*mulfak,d5

Tab:    move.l  d5,(a0)+
        add d6,d5
        move.l  #100*mulfak,(a1)+
        dbra    d7,Tab

Wait_Rt:move.l  4(a6),d0
        and.l   #$000fff00,d0
        cmp.l   #$00000f00,d0
        bne.s   Wait_Rt
        btst    #10,$16(a6)
        bne NoModify

        move    #800*mulfak,d2
        move    maus+2(pc),d0
        sub #124,d0
        mulu    #2*mulfak,d0
        lea x+4(pc),a0
        lea y(pc),a1
        move.l  d0,d3
        move    #anzahl-1,d7

Approx: move.l  (a0)+,d1
        sub d1,d0
        bpl.s   Positiv
        neg d0

Positiv: cmp    d0,d2
        bmi.s   Found
        move    d0,d2
        move    d3,d0
        addq.l  #4,a1
        dbra    d7,Approx
        sub #2*mulfak,d3
        bra.s   Last

Found:  subq.l  #4,a0
        add #2*mulfak,d3
        cmp.l   (a0),d3
        bpl NoSprite
        sub #4*mulfak,d3
Last:   cmp.l   -8(a0),d3
        bmi NoSprite
        sub #2*mulfak,d3
        move.l  d3,-(a0)
        move    maus(pc),d0
        sub #57,d0
        mulu    #mulfak,d0
        subq    #2,d0
        move    d0,2(a1)
        move    #mulfak*mulfaktor,d4
        bsr MakeSpline

        lea $dff000,a6
        lea bild(pc),a0
        move.l  (a0)+,a4
        move.l  (a0),d1
        move.l  a4,(a0)
        move.l  d1,-(a0)
        move.l  a4,a5
        bsr ClearPic
        lea x+4*anzahl(pc),a0
        move.l  (a0),d0

Draw:   move.l  d0,-(sp)
        move    #mulfak*mulfaktor,d1
        bsr Funktionswert           ; y-Wert in d1
        divu    #mulfak,d1
        move.l  (sp),d0
        move.l  d0,d4
        divu    #mulfak,d0
        ext.l   d0
        lea lineto(pc),a1
        move    (a1)+,d2
        move    (a1)+,d3
        move    d1,-(a1)
        move    d0,-(a1)
        tst d1
        bmi.s   NoLine
        cmp #245,d1
        bpl.s   NoLine
        tst d3
        bmi.s   NoLine
        cmp #245,d3
        bpl.s   NoLine
        cmp.l   x+4*anzahl(pc),d4
        beq.s   NoLine
        lea mulu88(pc),a0
        lea 978(a4),a5
        bsr Line

NoLine: move.l  (sp)+,d0
        sub.l   #3*mulfak,d0
        cmp.l   x+4(pc),d0
        bge.s   Draw
        lea punkte(pc),a0
        moveq   #anzahl-1,d7

Clr_pk: move    (a0)+,d0
        move.l  (a0)+,a1
        bclr    d0,(a1)
        dbra    d7,Clr_pk
        lea mulu88(pc),a0
        lea x+4(pc),a1
        lea y+4(pc),a2
        lea punkte(pc),a3
        moveq   #anzahl-1,d7

Knoten: move.l  (a1)+,d1
        divu    #mulfak,d1
        move.l  (a2)+,d2
        divu    #mulfak,d2
        add d2,d2
        lea bs+978(a4),a5
        add (a0,d2.w),a5
        move    d1,d2
        lsr #3,d2
        lea (a5,d2.w),a5
        moveq   #15,d2
        and.b   d2,d1
        not.b   d1
        bset    d1,(a5)
        move    (a3),d0
        move    d1,(a3)+
        move.l  (a3),d1
```

»Splines.asm«: Berechnet Kurven (Anfang)

$$1-w = (x_{j+1}-x)(x_{j+1}-x_j) \quad ; \quad w\varepsilon[0,1],$$

$$s_j(w) = wy_{j+1}+(1-w)y_j+(x_{j+1}-x)^2((w^3-w)m_{j+1}-((1-w)^3-(1-w))m_j)/6$$

Nunmehr ist jede Spline-Funktion – bezüglich w – im Intervall zwischen Null und Eins definiert. Von Interesse sind hauptsächlich die Randpunkte Null und Eins, und in diesen Punkten hat entweder w oder (1-w) den Wert Null. Um nach m aufzulösen, müßen die ersten Ableitungen der Abschnitte des Splines in den Randpunkten gleichgesetzt werden. Die erste Ableitung der obigen Gleichung lautet:

$$s(w)/dx = s'(w) = (y_{j+1}-y_j)/(x_{j+1}-x_j) + (x_{j+1}-x_j)((3w^2-1)m_{j+1}+(3(1-t)^2-1)m_j)/6$$

Wenn wir nun s'i+1(1)=s'i(0) setzten, ergibt sich folgendes System aus N-2 Gleichungen:

$$(x_i-x_{i-1})m_{i-1}+2(x_{i+1}-x_{i-1})m_i+(x_{i+1}-x_i)m_{i+1} = 6((y_{i-1}-y_i)/(x_{i-1}-x_i) - (y_j-y_{i-1})/(x_i-x_{i-1}))$$

Indem wir die Beziehung

$$D_i = x_{i+1}-x_i \ , \ di = 2(x_{i+1}-x_{i-1})$$

und

$$a_i = 6((y_{i+1}-y_i)/(x_{i+1}-x_i)-(y_j-y_{i-1})/(x_i-x_{i-1}))$$

einführen, erhalten wir z.B. für fünf Knoten das folgende Gleichungssystem:

$$\begin{pmatrix} \delta_2 & \Delta_2 & 0 \\ \Delta_2 & \delta_3 & \Delta_3 \\ 0 & \Delta_3 & \delta_4 \end{pmatrix} \begin{pmatrix} m_2 \\ m_3 \\ m_4 \end{pmatrix} = \begin{pmatrix} \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix}$$

Hierbei stimmt die Diagonale unter der Hauptdiagonalen mit der Diagonale über der Hauptdiagonalen überein (Symmetrisches tridiagonales System).

Folgend wollen wir nun die Implementation (»Spline.asm«) betrachten. Hierbei ist zu erwähnen, daß es uns nicht darauf ankam, ein absolut bedienerfreundliches Programm zu generieren. Überdies sollte man Verbesserungen und Erweiterungen ausprobieren, da so der Einstieg in das Programm einfacher ist. Zum Ansporn kann das Programm auf folgende Funktionen erweitert werden (nach Schwierigkeitsgrad steigend):

– das Koordinatensystem ein-/ausschalten (in der Copper-Liste mit »bchg« zwischen $a200 und $b200 umschalten)

– die Anzahl der Knoten variabel modifizieren

```
        move.l  a5,(a3)+
        move.l  d1,a5
        bclr    d0,(a5)
        dbra    d7,Knoten
        move.l  copperBase,a0
        lea Copper-sprite(a0),a0
        move.l  a4,d0
        move    d0,6(a0)
        swap    d0
        move    d0,2(a0)
        swap    d0
        add.l   #bs,d0
        move    d0,14(a0)
        swap    d0
        move    d0,10(a0)

NoModify:
        lea $a(a6),a0
        lea maus(pc),a2
        move    (a2)+,d1
        move    (a2)+,d0
        cmp #126,d0
        bpl.s   Xmin
        move    #126,-2(a2)
        bra.s   NoMaus

Xmin:   cmp #436,d0
        bmi.s   Xmax
        move    #435,-2(a2)
        bra.s   NoMaus
Xmax:   cmp #57,d1
        bpl.s   Ymin
        move    #57,-4(a2)
        bra.s   NoMaus
Ymin:   cmp #299,d1
        bmi.s   Ymax
        move    #298,-4(a2)
NoMaus: move    (a0),(a2)
        bra.s   NoSprite

Ymax:   move.b  (a0)+,d1
        sub.b   (a2)+,d1
        ext d1
        add d1,-5(a2)
        move.b  (a0)+,d0
        sub.b   (a2)+,d0
        ext d0
        add d0,-4(a2)
        move    -(a0),-(a2)
        move    -(a2),d0
        move    -(a2),d1
        moveq   #0,d3
        moveq   #3,d2
        move.l  copperBase,a0
        move.b  d1,(a0)
        btst    #8,d1
        beq.s   noY
        bset    #2,d3
noY:    add d2,d1
        move.b  d1,2(a0)
        btst    #8,d1
        beq.s   noZ
        bset    #1,d3
noZ:    lsr #1,d0
        bcc.s   noX
        bset    #0,d3
noX:    move.b  d0,1(a0)
        move.b  d3,3(a0)            ; Sprite setzen

NoSprite:
        btst    #6,$bfe001
        bne Wait_Rt
```

```
        move.l  4.w,a6
        lea GrafikName(pc),a1
        moveq   #0,d0
        jsr OpenLibrary(a6)
        tst.l   d0
        beq.s   NoGrafik
        move.l  d0,a1
        lea $dff000,a5
        move.l  38(a1),$80(a5)
        clr $88(a5)
        move    #$8020,$96(a5)
        jsr CloseLibrary(a6)

NoGrafik:
        jsr Permit(a6)
        move.l  copperBase(pc),a1
        move.l  #Copper_Ende-sprite+5*bs,d0
        jsr FreeMem(a6)
        lea x+4(pc),a1
        lea y+4(pc),a2
        moveq   #anzahl-1,d7

Change: move.l  (a1),d0
        sub #mulfak*x_achse,d0
        muls    #10*mulfaktor/mulfak,d0
        divs    #x_delta,d0
        ext.l   d0
        move.l  d0,(a1)+
        move.l  (a2),d0
        neg d0
        add #mulfak*y_achse,d0
        muls    #10*mulfaktor/mulfak,d0
        divs    #y_delta,d0
        ext.l   d0
        move.l  d0,(a2)+
        dbra    d7,Change
        move    #10*mulfaktor,d4
        bsr MakeSpline

        pea m(pc)
        pea x(pc)
        pea y(pc)
        moveq   #anzahl-2,d7
        lea DosName(pc),a1
        moveq   #0,d0
        move.l  4.w,a6
        jsr OpenLibrary(a6)
        move.l  d0,a6
        beq Ende
        jsr Output(a6)
        move.l  d0,handle

Funktion:
        move    d7,-(sp)
        addq.l  #4,10(sp)
        addq.l  #4,6(sp)
        addq.l  #4,2(sp)
        move.l  2(sp),a2
        move.l  6(sp),a1
        move.l  10(sp),a0
        bsr Trans
        lea koeffizient(pc),a0
        move    d0,(a0)+
        move    d3,(a0)+
        move    d5,(a0)+
        move    d4,(a0)+
        move    2(a1),(a0)+
        move    6(a1),(a0)
        move    d6,-(sp)
        clr.l   laenge
        lea koeffizient(pc),a1
        lea format(pc),a0
```

```
        lea buffer(pc),a3
        lea Routine(pc),a2
        move.l  4.w,a5
        jsr RawDoFmt(a5)
        move    (sp)+,d6
        move.l  laenge(pc),d3
        lea buffer(pc),a0

Form:   move.b  (a0)+,d0
        beq.s   FormOk
        cmp.b   #"0",d0
        bmi.s   Form
        cmp.b   #"9",d0
        bgt.s   Form
        lea -1(a0),a2

Zahl:   cmp.b   #"0",(a0)+
        bmi.s   Komma
        cmp.b   #"9",-1(a0)
        ble.s   Zahl
Komma:  lea -1(a0),a1
        move    d6,d7
Stelle: cmp.b   #" ",-(a1)
        bne.s   NoNull
        move.b  #"0",(a1)
NoNull: dbra    d7,Stelle
        move.l  a2,a3
        cmp.b   #"-",-(a3)
        bne.s   Space
        move.b  #"0",(a3)
Sign:   cmp.b   #"+",-(a3)
        bne.s   Sign
        move.b  #"-",(a3)
Space:  cmp.l   a2,a1
        bmi.s   NoSpace
        move.b  (a2)+,-2(a2)
        bra.s   Space
NoSpace:move.b  #".",-(a1)
        cmp.b   #" ",-(a1)
        bne.s   Form
        move.b  #"0",(a1)
        bra.s   Form

FormOk: move.l  handle,d1
        move.l  #buffer,d2
        jsr Write(a6)
        move    (sp)+,d7
        dbra    d7,Funktion
        lea 12(sp),sp
        move.l  handle,d1
        move.l  #format2,d2
        move.l  #format3-format2,d3
        jsr Write(a6)
        pea m+2(pc)
        clr -(sp)
        moveq   #anzahl-2,d7

Moment: move    d7,-(sp)
        addq.l  #4,4(sp)
        addq.w  #1,2(sp)
        clr.l   laenge
        lea koeffizient(pc),a1
        move    2(sp),(a1)
        move.l  4(sp),a0
        move    (a0),2(a1)
        lea format3(pc),a0
        lea buffer(pc),a3
        lea Routine(pc),a2
        move.l  4.w,a5

        jsr RawDoFmt(a5)
        move.l  handle,d1
```

»Splines.asm«: (Fortsetzung)

```
        move.l  #buffer,d2
        move.l  laenge(pc),d3
        jsr Write(a6)
        move    (sp)+,d7
        dbra    d7,Moment
        addq.l  #6,sp
        move.l  a6,a1
        move.l  4.w,a6
        jsr CloseLibrary(a6)

mm:     btst    #10,$dff016
        bne.s   mm

Ende:   clr.l   d0
        rts

Routine:move.b d0,(a3)+
        addq.l  #1,laenge
        rts

Trans:
        lea x2(pc),a3
        lea y2(pc),a4
        lea m2(pc),a5
        moveq   #1,d7

Move10: move.l  (a0)+,d0
        muls    #10,d0
        move.l  d0,(a5)+
        move.l  (a1)+,d0
        muls    #10,d0
        move.l  d0,(a3)+
        move.l  (a2)+,d0
        muls    #10,d0
        move.l  d0,(a4)+
        dbra    d7,Move10
        lea x2(pc),a1
        lea y2(pc),a2
        lea m2(pc),a0
        move    #100*mulfaktor,-(sp)

Over:   move    (sp),d0
        ext.l   d0
        divu    #10,d0
        move    d0,(sp)

        move.l  (a0),d0
        divs    #10,d0
        ext.l   d0
        move.l  d0,(a0)
        move.l  4(a0),d0
        divs    #10,d0
        ext.l   d0
        move.l  d0,4(a0)
        move.l  (a1),d0
        divs    #10,d0
        ext.l   d0
        move.l  d0,(a1)
        move.l  4(a1),d0
        divs    #10,d0
        ext.l   d0
        move.l  d0,4(a1)
        move.l  (a2),d0
        divs    #10,d0
        ext.l   d0
        move.l  d0,(a2)
        move.l  4(a2),d0
        divs    #10,d0
        ext.l   d0
        move.l  d0,4(a2)
        move.l  (a0),d0
        sub 6(a0),d0
        muls    (sp),d0
        move.l  (a1),d1
        sub 6(a1),d1
        divs    d1,d0
        ext.l   d0

        move.l  (a1),d2
        muls    d0,d2
        divs    (sp),d2
        move.l  (a0),d3
        sub d2,d3
        asr #1,d3
        divs    #6,d0
        move.l  (a1),d2
        move    6(a1),d4
        add d4,d2
        muls    d1,d2
        muls    d4,d4
        divs    (sp),d2
        move    d2,d5
        muls    d3,d2
        divs    (sp),d4
        bvs Over
        add d4,d5
        move    d5,d6
        muls    2(a1),d5
        muls    6(a1),d4
        divs    (sp),d5
        bvs Over

        move    d5,d7
        divs    (sp),d4
        bvs Over

        sub d4,d5
        muls    d0,d5
        add.l   d5,d2
        move.l  (a2),d5
        sub 6(a2),d5
        muls    (sp),d5
        sub.l   d2,d5
        divs    d1,d5
        move    2(a2),d1
        muls    (sp),d1
        muls    d0,d7
        muls    d3,d6
        add.l   d7,d6
        move    2(a1),d7
        muls    d5,d7
        move    2(a2),d4
        muls    (sp),d4
        sub.l   d6,d4
        sub.l   d7,d4
        divs    (sp),d4
        move    (sp)+,d7
        moveq   #-2,d6

Stell:  addq    #1,d6
        ext.l   d7
        divu    #10,d7
        bne.s   Stell
        rts

ClearPic:
        btst    #14,2(a6)
        bne.s   ClearPic
        move.l  a5,$54(a6)
        clr $66(a6)
        move.l  #$1000000,$40(a6)
        move    #300*64+44,$58(a6)
        rts

maus:   dc.w    152,285,0

Init:
        btst    #14,2(a6)
        bne.s   Init
        move    #-1,$72(a6)
        move.l  #-1,$44(a6)
        move    #breite,$60(a6)
        move    #$8000,$74(a6)

        move    #255,d7
        lea mulu88(pc),a0
        moveq   #0,d0

mlo:    move    d0,(a0)+
        add #breite,d0
        dbf d7,mlo
        rts

mulu88:dcb.w    256,0

MakeSpline:
        lea x(pc),a0
        lea y(pc),a1
        lea 4(a0),a4
        moveq   #anzahl-2,d7
        lea delta(pc),a2
        lea 4(a2),a3

Loop:   move.l  4(a4),d0
        sub.l   (a4)+,d0
        move.l  d0,(a3)+
        dbf d7,Loop

        lea delta2+8(pc),a3
        lea 4(a0),a4
        lea 8(a1),a5
        addq.l  #4,a2
        lea alpha+8(pc),a6
        moveq   #anzahl-3,d7

Loop2:  move.l  8(a4),d0
        sub.l   (a4)+,d0
        add.l   d0,d0
        move.l  d0,(a3)+
        move.l  4(a5),d0
        sub.l   (a5),d0
        move.l  4(a2),d1
        muls    d4,d0
        divs    d1,d0
        ext.l   d0
        move.l  (a5)+,d1
        sub.l   -8(a5),d1
        move.l  (a2)+,d2
        muls    d4,d1
        divs    d2,d1
        ext.l   d1
        sub.l   d1,d0
        add.l   d0,d0
        move.l  d0,d1
        add.l   d0,d0
        add.l   d1,d0
        move.l  d0,(a6)+
        dbf d7,Loop2

        lea m(pc),a6
        clr.l   (a6)
        clr.l   4*anzahl(a6)
        lea alpha+8(pc),a0
        lea delta+8(pc),a1
        lea delta2+8(pc),a2
        moveq   #anzahl-4,d7

Loop3:  move.l  (a0)+,d0
        move.l  (a1)+,d1
        move.l  (a2)+,d2
        muls    d1,d0
        divs    d2,d0
        ext.l   d0
        sub.l   d0,(a0)
        muls    d1,d1
        divs    d2,d1
        ext.l   d1
        sub.l   d1,(a2)
        dbf d7,Loop3
        lea alpha+anzahl*4(pc),a0
        lea delta+anzahl*4(pc),a1
        lea delta2+anzahl*4(pc),a2
        lea m+anzahl*4(pc),a3
        moveq   #anzahl-3,d7

Loop4:  move.l  -(a0),d0
        move.l  -(a1),d1
        move.l  -(a2),d2
        move.l  (a3),d3
        muls    d3,d1
        muls    d4,d0
        sub.l   d1,d0
        divs    d2,d0
        ext.l   d0
        move.l  d0,-(a3)
        dbf d7,Loop4
        rts

Funktionswert:
        move    d1,-(sp)
        lea x+8(pc),a0
        lea y(pc),a1
        lea delta(pc),a2
        lea m(pc),a3

Suchen: addq.l  #4,a1
        addq.l  #4,a2
        addq.l  #4,a3
        cmp.l   (a0)+,d0
        bgt.s   Suchen
        move.l  (a1)+,d1
        move.l  (a2),d2
        move.l  (a3)+,d3
        sub.l   -8(a0),d0
        muls    (sp),d0
        divs    d2,d0
        ext.l   d0
        move.l  (a1),d4
        move.l  (a3),d5
        muls    d0,d4
        moveq   #0,d6
        move    (sp),d6
        sub.l   d0,d6
        muls    d6,d1
        add.l   d4,d1
        muls    d2,d2
        divs    (sp),d2
        ext.l   d2
        move.l  d0,d4
        muls    d0,d0
        divs    (sp),d0
        ext.l   d0
        muls    d4,d0
        muls    (sp),d4
        sub.l   d4,d0
        divs    (sp),d0
        ext.l   d0
        muls    d0,d5
```

»Splines.asm«: (Fortsetzung)

```
        move.l  d6,d4
        muls    d6,d6
        divs    (sp),d6
        ext.l   d6
        muls    d4,d6
        muls    (sp),d4
        sub.l   d4,d6
        divs    (sp),d6
        ext.l   d6
        muls    d6,d3
        add.l   d5,d3
        divs    (sp),d3
        ext.l   d3
        muls    d3,d2
        moveq   #1,d3
        moveq   #6,d0

Again:  divs    d0,d2
        bvc.s   NoError
        mulu    #10,d0
        mulu    #10,d3
        bra.s   Again

NoError:ext.l   d2
        muls    d3,d2
        add.l   d2,d1
        divs    (sp),d1
        ext.l   d1
        addq.l  #2,sp
        rts

Koordinatensystem:
        leabs+978(a4),a3
        leabs(a3),a3
        moveq   #0,d0
        move    #y_achse,d1
        move    d1,d3
        move    #620,d2
        move.l  a3,a5
        bsrLine
        move    #x_achse,d0
        move    d0,d2
        moveq   #0,d1
        move    #240,d3
        move.l  a3,a5
        bsrLine
        move    #x_delta,d0
        moveq   #0,d1
        move    #y_delta,d2

X_Punkt:moveq   #0,d3
Y_Punkt:move    d3,d4
        addd4,d4
        move.l  a3,a5
        add(a0,d4.w),a5
        move    d1,d4
        move    d1,d5
        lsr#3,d4
        lea(a5,d4.w),a5
        moveq   #15,d4
        and.b   d4,d5
        not.b   d5
        bset    d5,(a5)
        addd2,d3
        cmp#240,d3
        ble.s   Y_Punkt
        addd0,d1
        cmp#620,d1
        ble.s   X_Punkt
        rts

Line:   cmpd1,d3
        bgt.s   nohi
        exgd0,d2
        exgd1,d3

nohi:   move    d0,d4
        move    d1,d5
        addd5,d5
        add(a0,d5.w),a5
        lsr#4,d4
        addd4,d4
        lea(a5,d4.w),a5
        subd0,d2
        subd1,d3
        moveq   #15,d5
        and.l   d5,d0
        move    d0,d4
        ror.l   #4,d0
        eord5,d4
        moveq   #0,d5
        bset    d4,d5
        move    #4,d0
        tstd2
        bpl.s   11
        addq.w  #1,d0
        negd2

11:     cmpd2,d3
        ble.s   12
        exgd2,d3
        subq.w  #4,d0
        addd0,d0
12:     move    d3,d4
        subd2,d4
        addd4,d4
        addd4,d4
        addd3,d3
        moveq   #0,d6
        move    d3,d6
        subd2,d6
        bpl.s   13
        moveq   #16,d1
        or d1,d0
13:     addd3,d3
        addd0,d0
        addd0,d0
        addq.w  #1,d2
        lsl#6,d2
        addq.w  #2,d2
        swap    d3
        move    d4,d3
        or.l    #$0bca0001,d0

wbl:    btst    #14,2(a6)
        bne.s   wbl
        move.l  d3,$62(a6)
        move    d6,$52(a6)
        move.l  a5,$48(a6)
        move.l  a5,$54(a6)
        move.l  d0,$40(a6)
        move    d2,$58(a6)
        rts

lineto:dc.w     0,0
punkte:dcb.w    3*anzahl,0
y:      dcb.l   anzahl+1,0
delta:  dcb.l   anzahl+1,0
delta2:dcb.l    anzahl+1,0
alpha:  dcb.l   anzahl+1,0
m:      dcb.l   anzahl+1,0              ; Momente
x:      dcb.l   anzahl+1,0
x2:     dc.l    0,0
y2:     dc.l    0,0
m2:     dc.l    0,0
laenge:dc.l     0
buffer:dcb.w    50,0

koeffizient:    dcb.w7,0
copperBase:     dc.l 0
bild:   dc.l    0
bild2:  dc.l    0
handle:dc.l     0

format:
 dc.b "s(x) = +%7dx",179,"+%7dx",178,
 dc.b "+%7dx+%7d,   ", "+%7d<= x >=+%7d",10,0

format2:dc.b    10,"Spline - Momente:",10,0
format3:dc.b    "m[%d] = %5d / 1000",10,0

DosName:   dc.b"dos.library",0
GrafikName:     dc.b "graphics.library",0

        EVEN

sprite:dc.l     0
        dc.w    %0010000000000000,0
        dc.w    %0101000000000000,0
        dc.w    %0010000000000000,0
        dc.l    0

Copper:
 dc.w $e0,0,$e2,0,$e4,0,$e6,0,$e8,0,$ea,0
 dc.w $120,0,$122,0
 dc.w $100,$b200,$108,0
 dc.w $8e,$3081,$90,$30c1,$92,$28,$94,$d0
 dc.w $180,0,$182,$f,$184,$f00,$186
 dc.w $f00,$1a2,$f00
 dc.w $188,$f0,$18a,$f,$18c,$f00,$18e,$f00
 dc.w $124,0,$126,0,$128,0,$12a,0,$12c,0
 dc.w $12e,0,$130,0
 dc.w $132,0,$134,0,$136,0,$138,0,$13a,0
 dc.w $13c,0,$13e,0
 dc.l -2
Copper_Ende:
```


»Splines.asm«: Unser Programm, um Kurven durch gegebene Punkte zu berechnen (Devpac 2.0)

(Konstante »anzahl« löschen und mit einer Speicherstelle arbeiten)
– das Koordinatensystem variabel modifizieren (ähnlich wie bei den Knoten)
– während der Bestimmung der Spline-Funktion, die Anzahl der Knoten verändern (variable Anzahl der Knoten verwenden)
– die Spline-Funktion speichern/laden (benötigt wird der Bereich m (Momente) und die Anzahl der Knoten)

Erweiterungen erwünscht

Programm-Dokumentation:

Beim Start des Programms erscheint auf dem Bildschirm in verschiedenen Farben: ein Koordinatensystem, eine Gerade und ein kleines Kreuz. Das kleine Kreuz fungiert als Mauszeiger. Beim Betätigen der rechten Maustaste wird ein Knoten des Splines zum Mauszeiger positioniert. Hieran sollte erwähnt werden, daß die Spline-Funktion blau und die einzelnen Knoten rot sind. Die Verwaltung der Knoten erfolgt auf einer dynamischen Art und Weise, d.h. der Knoten, der in der Nähe liegt, wird verwendet; hinzu wird noch darauf geachtet, daß sich die einzelnen Knoten nicht überschneiden. Schließlich wird durch Betätigen der linken Maustaste das Programm beendet und die Spline-Funktion auf zwei Arten ausgegeben. Die erste Form ist die des kubischen Polynoms und in der zweiten sind die einzelnen Momente aufgezählt (genauere Werte). Zuletzt wird auf das Betätigen der rechten Maustaste gewartet.

Speziell zur Implementation:

Zunächst die Erklärung der einzelnen Konstanten: Die »anzahl« steht für die Anzahl der Knoten, die mindestens vier betragen muß. Durch »mulfak« und »mulfaktor« wird eine Pseudo-Gleitpunktdarstellung simuliert. Anhand »x_links«, »x_rechts«, »y_oben«, »y_unten«, »x_delta«, »y_delta«, »y_achse« und »x_achse« wird das kartesische Koordinatensystem generiert.

Im Programmabschnitt »MakeSpline« werden für die Spline-Funktion gebrauchten Momente berechnet und in »m« abgelegt. Mittels »Funktionswert« wird der benötigte Wert durch den Punkt (in d0) berechnet. Mit Hilfe von »Koordinatensystem« wird das Achsenkreuz abgebildet.

Zum Schluß wollen wir noch einige Vorteile der Spline-Interpolation gegenüber der Interpolation mit einer einzigen Polynomfunktion vom Grad N aufgreifen:
– die Splineberechnung erfolgt schneller, da weniger Unbekannte vorhanden sind
– die Splines wirken glätter, da sie geringe Felligkeit haben und geben daher den Verlauf einer Funktion besser wieder

Die Gesamtkrümmung der interpolierenden Kurve wird minimal – eine Eigenschaft, die ein durch die Knoten gelegtes elastisches Lineal ebenfalls annähernd realisiert. Damit haben wir ein mechanisches Modell für die Spline-Interpolation. ■

Buntgemischtes für Programmierer

Tips & Tricks und tolle Tools

Die Rubrik Tips & Tricks ist eine der beliebtesten des AMIGA-Magazins. und auch im Sonderheft »Faszination Programmieren« darf sie natürlich nicht fehlen. Auf den folgenden Seiten Sie alle erdenklichen Hilfen und Kniffe, wie Sie besser und schneller mit dem Amiga umgehen, und wie Sie – speziell bei der Programmierung – die eine oder andere Hürde umschiffen.

Was sind überhaupt »Tools«? Wörtlich übersetzt heißt das »Werkzeuge«. Gemeint sind all die kleinen – oder manchmal auch größeren – Programme, mit denen Sie z.B. Texte konvertieren, Festplatten aufräumen, nach Dateien suchen etc. Also all die kleinen Helfer, die Ihnen die Arbeit mit dem Amiga erleichtern.

Auf den folgenden Seiten finden Sie eine Menge solcher Tools und viele kleine Tips & Tricks als Hilfe für Ihr Programmiererleben.

Wir fangen an mit buntgemischten Kniffen rund um den Amiga. Ab Seite 41 finden Sie im Rahmen unserer Knobelecke Tricks, wie Sie Programme schneller machen und wie Sie überhaupt an Programmieraufgaben herangehen, und von Seite 51 bis 60 geben dann die Tools den Ton an. Schauen Sie sich um, es sollte auch für Sie etwas dabei sein.

Wenn Sie selbst ein paar Helferlein programmiert haben – oder den einen oder anderen Trick für den Amiga ausbaldowert haben, schicken Sie Ihren Beitrag an die Redaktion »Faszination Programmieren«, d.h. an:

AMIGA-Redaktion
»Faszination Programmieren«
Markt & Technik Verlag
Hans-Pinsel-Str. 2
85540 Haar bei München

In den nächsten Sonderheften werden wir die besten Einsendungen veröffentlichen, Wieder bunt gemischt, so daß für jeden etwas dabei ist. Die nächste Ausgabe von »Faszination Programmieren« ist für Anfang 1994 geplant.

Für alle veröffentlichten Beiträge gibt´s ein Honorar. Falls der Platz für den Abdruck aller ausgewählten Beiträge nicht reicht, drucken wir evtl. ein paar Einsendungen im AMIGA-Magazin ab. Also: Es lohnt sich mitzumachen.

Und dann noch eine Bitte und ein Dank: Dank allen Programmierer, die sich an den Tips & Tricks beteiligt haben. Und die Bitte um Geduld, wenn ihr Beitrag nicht veröffentlicht wurde. Im nächsten Sonderheft sollte es klappen.

Und noch eine Bitte: Wir haben so viele Tips, die wir am liebsten alle drucken würden, aber was für Tips bevorzugen Sie? Kurze, kleine? Oder ausführliche? BASIC- oder C-, ARexx oder Assembler-Tips? Schreiben Sie uns, damit wir Ihr nächstes »Faszination Programmieren« noch besser machen können.

Kleine Pause mit DELAY

Mit dem DOS-Befehl WAIT kann man Sekunden oder Minuten warten. Wenn es aber genauer sein soll, benutzt man DELAY (ab OS 2.0). Als Parameter gibt man die Wartedauer in 50stel Sekunden als Dezimalzahl an.

Verwendungszweck ist z.B. eine Batch-Datei in der Programme mit dem Befehl RUN gestartet werden. Nach RUN wartet man (die Zeit muß man abschätzen), um dem »Laufwerksgesäge« ein Ende zu bereiten. DELAY ist residentfähig. *Thies Wellpott*

Fenster löschen...

Es gibt mehrere Wege, den Inhalt eines CLI Fensters zu löschen. Eine davon funktioniert über den Befehl ECHO: Man schreibt z.B. in der »Startup-Sequence« folgende Zeile:

```
echo <Ctrl L>
```

Beim »L« sollten jetzt die Vorder- und Hintergrundfarben vertauscht sein.

Auch vom CLI kann man <Ctrl L> anwenden, aber dafür erscheint das häßliche »...Unknown command«. *Thomas Binggeli*

IsResident

Möchte man in einer Batch-Datei abfragen, ob ein Programm resident ist, schreibt man ab jetzt – d.h. ab OS 2.0. – nur noch:

```
IsResident Name
```

Ist »Name« nicht resident wird WARN (=5) als Rückkehrcode gesetzt, sonst 0. IsResident ist residentfähig, *Thies Wellpott*

FastFileSystem für Disketten

Unter Kickstart/Workbench 2.0 wird jetzt auch das FastFileSystem für Disketten voll unterstützt. Eine entsprechende Diskette wird automatisch als solche erkannt (Das SHELL-Kommando INFO zeigt dann 879 und nicht nur 837 KByte an). Dazu muß beim Formatieren nur die Options FFS mit angegeben werden.

Programme laufen wie der Blitz

Wer viel mit Programmen wie MandelBlitz oder MAK zur Berechnung von Mandelbrotgrafiken etc. mit der HiRes-Interlaced Auflösung arbeitet, sollte den Workbenchscreen nach vorne holen. Das mit der Tastaturkombination <Amiga_links M> machbar. Das ergibt teilweise Geschwindigkeitssteigerungen um bis zu 150 Prozent.

Die ganze Sache kann man zusätzlich beschleunigen, indem man die Anzahl der Bitplanes auf 1 stellt (z.B. mit WB_Planes). Dadurch gewinnt man nochmals ca. 5% Zeit. Die angegebenen Werte wurden in diesem Fall auf einem normalen Amiga 2000 (1 MByte ChipRAM) gemessen.

Wenn Ihr Rechner über FastRAM verfügt, fällt der Geschwindigkeitsgewinn etwas geringer aus.

Die erwähnten Programme befinden sich auf denen im folgenden genannten Public-Domain-Disketten *Thomas Binggeli*

Quellennachweis:
- MandelBlitz 1.0 — Fish Disk 378
- MAK — Fish Disk 522
- WB_Planes — Fish Disk 543

Geschwindigkeitsvergleich

Screen Res. v. WB.	Aktueller Screen	Zeit für Standard Mandelbrot
Hires (640x512)	MandelBlitz 640x512	ca 13 min 05 sek
Hires (640x512)	WB 1 640x256	ca 5 min 30 sek
Hires (640x512)	WB 2 640x256	ca 5 min 13 sek

*** WB 1 = WB Screen mit 1 Plane : WB 2 = WB Screen mit 2 Planes



Beim Formatieren von der Workbench aus wird einem diese Wahl leider nicht gelassen. Auf den Diskdoctor sollte man aber besser verzichten, er zerstört die Diskette.

Vorteile des FFS:

Neben den etwas 40 KByte mehr Speicher für Ihre Listings und Programme wird der Zugriff auf Disketten auch etwas beschleunigt.

Nachteile:

Die Sache hat nur einen Haken: Amiga-Besitzer, die (noch) nicht Kickstart/Workbench 2.0 besitzen, dürften leichte Probleme haben, diese Disketten zu lesen, d.h. sie können Ihre in der beschriebenen Art präparierten Disketten nicht an Besitzer älterer Amiga-Modelle weitergeben. *Rüdiger Dreier*

FFS: FORMAT und INSTALL (ab OS 2.0)

Formatiert man eine Diskette von der Workbench aus, oder ohne Optionen aus der Shell, bekommt die Diskette das Format des normalen File-Systems (OFS).

Um nicht jedesmal in der Shell beim FORMAT-Befehl für das Fast-File-System ewig lange Optionen angeben zu müssen, gibt´s einen Trick. Die »Shell-Startup« im Verzeichnis »S:« muß nur um folgende Zeilen ergänzt werten:

```
alias fformat "FORMAT DRIVE [] NAME
                        EmptyFFS FFS NOICONS"
alias ffinstall "INSTALL DRIVE [] FFS"
```

Nun wird eine Diskette mit dem Befehl

```
FFORMAT <laufwerk>
```

im Fast-File-Format mit Namen »EmptyFFS« formatiert (durch die Option NOICONS wird der Mülleimer nicht mitkopiert). Der Befehl

```
FFINSTALL <laufwerk>
```

dient dazu, die Diskette mit dem Fast-File-System Bootblock zu beschreiben.

Oberbuchner Christian

Preferences bei Midi-Software

Sollten Probleme bei der Benutzung von Midi-Software auftreten, indem bei der Datenübertragung zwischen Amiga und Midi-Keyboard u.a. die Maus-Geschwindigkeit stark zurückgeht und ein weiteres Arbeiten unmöglich wird, so könnte das an einer falschen Preferences-Einstellung liegen.

Die Übertragung von Midi-Daten erfolgt nämlich in einer Baudrate von 31 250. In den Preferences ist dagegen ein weit niedrigerer Wert voreingestellt, der für den Betrieb z.B. mit einem Modem ausgelegt ist. Versucht das Midi-Programm nun, Daten mit 31 250 Baud zu übertragen, können diese nicht mit der nötigen Geschwindigkeit tranferiert werden und es kommt zur geschilderten Erscheinung. Abhilfe schafft hier das Setzen der Baudrate auf 31 250 im Preferences-Teil »Serial«.

Christof Brühann

Debuggen von Befehlsdateien

Fehler in Befehlsdateien, auch Skript- oder Batch-Dateien genannt, sind schwer zu lokalisieren, da zur Fehlermeldung keine Zeilennummer ausgegeben wird. Bei Befehlsdateien, die selbst wieder Befehlsdateien erzeugen und ausführen (z.B. SPat), sind Fehler noch schwerer zu finden, da man nicht erfährt, in welcher Datei der Fehler auftrat. Besitzer der Workbench 2.0 können dem abhelfen, indem sie

```
Set Echo On
```

eingeben, bevor sie die fehlerhafte Befehlsdatei ausführen. Dadurch wird jeder CLI-Befehl vor der Ausführung ausgegeben. So kann man überprüfen, wo ein Fehler auftrat, und welcher Befehl welche Ausgabe erzeugt. Achtung: Vor der Ausgabe werden Befehlszeilen expandiert, d.h., Variablen und Parameter (wie $xyz und <Dir>) werden durch ihre Inhalte ersetzt. Auch der Escape-Character * wird expandiert.

Um die Ausgabe der Befehle wieder abzustellen, geben Sie »Unset Echo« oder »Set Echo Off« oder »Set Echo GrmblLFtx« ein.

Torsten Techmann

Noch ein Suchpfad gefällig?

Ein zusätzlicher Suchpfad kann im CLI oder der Shell mit

```
path xy add
```

eingebunden werden. Dies kann man natürlich bei jedem neuen Shell-Fenster eingeben, aber es gibt auch andere Möglichkeiten:

Beim Öffnen eines neuen Shell-Fensters wird (meist) die Datei »S:Shell-Startup« ausgeführt. Somit hat man den neuen Pfad in jedem so geöffneten Fenster zur Verfügung.

Nachteil: Einige Consolenfenster werden geöffnet, ohne daß »S:Shell-Startup« ausgeführt wird. Das ist z.B. der Fall, wenn man beim SAS-C Compiler V5.10 den Befehl LMK von der Workbench aus mit einem Doppelklick startet. Die in »S:Shell-Startup« angegebenen Pfade sind hier meist nicht vorhanden. Das ist gravierend, da der C-Compiler im Pfad gesucht wird, normalerweise befindet er sich aber im Verzeichnis LC und nicht in C:.

Abhilfe: In der Datei »S:Startup-Sequence« kann man bereits Pfade angeben. Sie müssen nur vor LOADWB stehen. Am besten bringt man Ergänzungen zur »Startup-Sequence« in »S:User-Startup« unter, die Datei wird beim Aufruf von LOADWB automatisch ausgeführt. Damit sind die Pfade in allen Consolenfenstern vorhanden, auch wenn »S:Shell-Startup« nicht ausgeführt wird. *Rüdiger Dreier*

PAL muß her

Sicher kennen Sie das: Von Zeit zu Zeit startet der AMIGA nicht im PAL-, sondern im NTSC-Modus. Als Folge hiervon fehlen die unteren 56 Bildschirmzeilen und manche Programme versagen sogar ihren Dienst. Was tun? Nochmals booten und wertvolle Zeit vergehen lassen oder den dezimierten Screen einfach erdulden?

Es geht auch anders: Mit dem Programm »PN_switch« von Disk A des »NewsFlash«-Diskettenmagazins Ausgabe 21 ist es möglich, nach dem Booten noch in den PAL-Modus umzuschalten. Das Programm befindet sich im »Prog«-Verzeichnis. Beim Aufruf im CLI gibt man als Parameter einfach »PAL« an. Das funktioniert allerdings nur, wenn der Computer mit einem der neueren AGNUS-Chips (8732 A ff.) ausgerüstet ist. *Sean Durkin*

Disketten aufräumen

Kennen Sie den Tip, um auf Disketten aufzuräumen und alle Dateien, die evtl. fragmentiert auf einer Diskette vorliegen, wieder zusammenhängend zu speichern, um wieder schnelleren Diskettenzugriff zu bekommen?

Mit zwei Diskettenlaufwerken ist´s besonders einfach: Einfach im CLI folgenden Befehl eingeben.

```
copy df0: to df1: all
```

Das kann aber bis zu 20 Minuten dauern und die Laufwerke laufen heiß. Wer über 2 MByte Speicher verfügt, kann das Ganze in ca. 4 Minuten über RAD: und RAM: mit einer Batch-Datei erledigen. Man kann so sogar auf die gleiche Diskette zurückgeschreiben. Oder auch auf eine andere, auf jeden Fall braucht man nur ein Laufwerk! Nennen Sie die Batch-Datei z.B. »DiskFix«. Dann sieht der Aufruf so aus:

```
execute DiskFix name
  ; "name" = der Name der neuen Diskette
```

Beachten Sie, daß in der »Startup-Sequence« der folgende Befehl stehen muß, damit das Ganze funktioniert:

```
setpatch r
```

Andernfalls müssen Sie den SETPATCH-Befehl zu Fuß eingeben. Hier die Batch-Datei zum einfachen Kopieren von Dateien und Aufräumen auf einer Diskette:

```
.key name
failat 11
mount rad:
delete ram:#? all
diskcopy df1: to rad: name "Rambo"
copy rad: ram: all
format drive rad: name "RAMBO" noicons
copy ram: rad: all
delete rad:t all
delete ram:#? all
diskcopy rad: to df1: name <name>
```

In der Mountlist muß die RAD: dasselbe Speichervermögen einer Diskette haben.

```
RAD:        Device = ramdrive.device
            Unit   = 0
            Flags  = 0
            Surfaces  = 2
            BlocksPerTrack = 11
            Reserved = 2
            Interleave = 0
            LowCyl = 0  ;  HighCyl = 79
            Buffers = 5
            BufMemType = 1
          #
```

Herbert Pittermann

Laufwerke raus »RemoveTrackdisk«

Das Programm »RemoveTrackdisk« entfernt alle im System befindlichen Laufwerke. Dadurch steht mehr Speicher zur Verfügung und außerdem wird das unangenehme »Laufwerksklicken« unterdrückt. Da nach Aufruf des Programms die Laufwerke nicht mehr benutzt werden können, ist das Utility hauptsächlich für Festplattenbesitzer ausgelegt.

```
/* RemoveTrackdisk - entfernt alle Laufwerke
   Aufruf mit Aztec V3.6:
         cc RemoveTrackdisk.c +l
         ln RemoveTrackdisk.o -lc32
         RemoveTrackdisk
*/
#include <functions.h>
struct Task *Task;
void main()
{while (Task=FindTask("trackdisk.device"))
                              /* Task suchen */
  RemTask(Task);              /* und entfernen */
}                                © 1993 M&T
```

»RemoveTrackdisk« bringt alle Diskettenlaufwerke zum Schweigen

»RemoveTrackdisk« findet mit Hilfe einer while-Schleife und dem Befehl

```
FindTask("trackdisk.device")
```

alle Tasks, welche die Laufwerke verwalten. Findet das Programm so einen Task, kann es ihn mit der Systemfunktion »RemTask()« entfernen. Die while-Schleife wird verlassen, wenn kein Trackdisk-Task mehr gefunden wird. *Christof Brühann*

C-Programme für 68000- bis 68030er

C-Programme, bei denen 4-Byte-Objekte auf einer durch vier teilbaren Adresse (4-Byte-Grenze) beginnen, laufen auf allen 680x0-Prozessoren; sie laufen allerdings auf 32-Bit-Maschinen schneller als Versionen, bei denen die Daten nicht auf 4-Byte-Grenzen ausgerichtet sind. Der Prozessor braucht im ersten Fall nur einen Speicherzugriff.

C-Compiler bieten eine Option, Code zu erzeugen (SAS-C: -l), in dem alle 4-Byte-Objekte auch an entsprechenden 4-Byte-Adressen abgelegt sind. Aber Achtung: Wenn Sie die Systemstrukturen benutzen, laufen einige Programme nicht mehr, da der Compiler automatisch innerhalb von Strukturen auch alle Komponenten auf 4-Byte-Grenzen erwartet, was natürlich nicht den Tatsachen entspricht, da das Betriebssystem platzsparend organisiert ist. Abhilfe schafft folgender kleiner Trick (SAS):

Binden Sie alle benötigten Includefiles als »Precompiled Headers« ein – beim Kompilieren dürfen Sie hier natürlich NICHT die -l Option angeben. Beim Übersetzen des eigentlichen Programms lassen Sie in der Kommandozeile ERST mit -H die vorkompilierte Datei einlesen und geben danach -l an.

Beim SAS-C hat die Reihenfolge der Parameter (besonder vor/hinter -H) eine Bedeutung: So fertigt »-d3 -H..« auch Debug-Informationen für Daten in der Datei »..« an, während »-H.. -d3« nur Daten für die im Programm eingelesenen Includes liefert. Die angegebene Reihenfolge interpretiert also alle Systemstrukturen normal, während alle neuen Programmdaten sauber auf 4-Byte-Grenzen gelegt werden.

Bei der Kombination von -w (16 Bit ints) und -l sollte man vorsichtig sein. Funktionen, die z.B. SHORT-Variablen gemischt mit anderen Variablen übergeben bekommen, erzeugen (V5.10) falschen Code. (siehe auch Schublade »Parameter«) *Rüdiger Dreier*

Keyboard-Device unter Kickstart V2.0

Bei Betriebssystemversionen vor Kickstart V2.0 enthielt das Keyboard-Device zwar schon die Funktion »KBD_READMATRIX«, allerdings ist sie erst ab Betriebssystem V2.0 fehlerfrei. Die Funktion hat die Aufgabe, den Status aller Tasten zu ermitteln. Man kann so den Zustand jeder beliebigen Taste bestimmen.

Das Listing »Keyboard_READMATRIX.c« zeigt, wie Sie die Funktion »KBD_READMATRIX« benutzen. Als Beispiel ermitteln wir hier den Zustand der <Caps Lock>-Taste und geben das Ergebnis aus. Nach dem bei der Device-Programmierung standardmäßigem Erstellen des Device-Blocks über »CreatePort()« und »CreateStdIO()« sowie nach Öffnen des Device mit »OpenDevice()« kann man den Device-Block für die spezielle Aufgabe vorbereiten. Die Funktion »KBD_READMATRIX«, welche im Device-Block unter

```
IOStdReq->io_Command
```

eingetragen wird, erwartet im Feld »io_Data« die Adresse eines Felds, in dem der Zustand der Tasten gespeichert werden soll. Die Länge des Felds tragen Sie dabei unter »io_Length« ein. Anschließend können Sie den Befehl durch Aufruf von »DoIO()« ausführen.

Nun ist zu klären, wie die Zustände der Tasten im Feld, das im Listing »array« heißt, gespeichert werden? Da zum Angeben des Zustands für jede Taste nur ein Bit benötigt wird, können in jedem Byte des Felds jeweils acht Zustände gespeichert werden.

Jedes gesetzte Bit steht für eine gedrückte Taste. Dabei ist das erste Byte für die Tasten mit den Codes 0 bis 7, das zweite für die Tasten-Codes 8 bis 15 usw. zuständig. Das zu untersuchende Byte im Feld läßt sich demnach mit »array[code/8]« angeben. Da dieses Byte den Zustand von acht Tasten angibt, ist eine AND-Verknüpfung mit dem Rest der Division des Tastencodes durch acht notwendig, um an den Zustand einer einzelnen Taste zu kommen.

Zur Vereinfachung haben wir im Listing das Makro »KEYSTATUS(code)« definiert, welches dem Programmierer mitteilt, ob die Taste mit dem Code »code« gedrückt ist oder nicht. So kann die Abfrage einfach mit

```
if (KEYSTATUS(CAPSCODE)==TRUE) ...
```

erledigt werden, wobei »CAPSCODE« der RAWKEY-Code der <Caps Lock>-Taste ist. Je nach Ergebnis dieser Abfrage wird entweder der Text »<Caps Lock> leuchtet« oder »<Caps Lock> leuchtet nicht« ausgegeben.

Christof Brühann

```
/*    Keyboard_READMATRIX - demonstriert die Keyboard-Device-Funktion READMATRIX
   Aufruf mit Aztec V3.6:    cc Keyboard_READMATRIX.c +l
                             ln Keyboard_READMATRIX.o -lc32
                             Keyboard_READMATRIX
                                (Kickstart V2.0 erforderlich)    */
#include <devices/keyboard.h>
#include <functions.h>
#define MOD8(a) (a-a/8*8)         /* Rest von Division durch acht */
#define KEYSTATUS(code) ((array[(code)/8])&(1<<(MOD8(code))) ? TRUE : FALSE)
                                  /* ermittelt Zustand einer Taste */
#define CAPSCODE 0x62             /* Code von <Caps Lock> */
#define LEN 16                    /* Länge des Feldes */
UBYTE array[LEN];
struct IOStdReq *IOStdReq;
struct MsgPort *MsgPort;
void main()
{if (MsgPort=(struct MsgPort *)CreatePort(0,0))          /* Device-Block erstellen */
 {if (IOStdReq=(struct IOStdReq *)CreateStdIO(MsgPort))  /* und Keyboard-Device öffnen */
  {if (!OpenDevice("keyboard.device",0,IOStdReq,0))
   {                                                    /* Device-Block beschreiben:*/
    IOStdReq->io_Command=KBD_READMATRIX;                /* Befehl eintragen */
    IOStdReq->io_Data=(APTR)&array;                     /* Adresse des Felds */
    IOStdReq->io_Length=LEN;                            /* Länge des Felds */
    DoIO(IOStdReq);               /* Befehl ausführen (Zustand aller Tasten ermitteln)*/
                                  /* dann speziell Zustand von <Caps Lock> prüfen > */
    if (KEYSTATUS(CAPSCODE)==TRUE) puts("<Caps Lock> leuchtet.");
     else puts("<Caps Lock> leuchtet nicht.");
    CloseDevice(IOStdReq);        /* Device schließen */
   }
   DeleteStdIO(IOStdReq);         /* Device-Block sowie Message-Port löschen */
  }
  DeletePort(MsgPort);
}}                   © 1993 M&T
```

»Keyboard_READMATRIX.c: Das Listing demonstriert, wie Sie jede Taste der Tastatur über das das Keyboard-Device kontrollieren



Standardfunktionen selbstgestrickt

Hier etwas für Tüftler, die ihre Programme gerne optimieren und daran herumfeilen: Sie können einige der Stringfunktionen der »c.lib« kinderleicht selbst implementieren.

❒ Ein Beispiel für eine Funktion zum Vergleichen von Strings:

```
void strcmp(s1,s2)
char *s1, *s2;
{  while(*s1++ = *s2++)
     ;
}
```

Eine Version von »strcmp«, bei der als Quell-String s2 und als Ziel s2 dient.

❒ Als nächstes eine Routine, um die Länge eines Strings zu ermitteln:

```
long strlen(s)
char *s;
{ long len=0;
   while(*s++) len++;
}
```

Die Funktion ist auch ziemlich leicht zu durchschauen. In ähnlicher Weise wie die beiden Funktionen lassen sich natürlich auch andere Stringbearbeitungsfunktionen selbst implementieren. Viel Spaß beim Experimentieren! *Chriastian Obergeschwandner*

Hardwaremäßige Joystick-Abfrage

Beim Amiga erfolgt die Abfrage von am Game-Port angeschlossenen Joysticks normalerweise über Devices, die eine multitaskinggerechte Ein- und Ausgabe ermöglichen. Doch bei manchen Programmen, z.B. Spielen, wird eine hohe Verarbeitungsgeschwindigkeit benötigt, die durch eine Device-Programmierung besonders bei der Joystick-Abfrage nicht immer gegeben ist. Soll außerden das Multitasking aus Geschwindigkeitsvorteilen sowieso ausgeschaltet werden, ist die hardwaremäßige Abfragung der Game-Ports nicht mehr zu vermeiden.

Das nächste Listing, »GetJoyData.C«, zeigt, wie man direkt über die Hardware-Register einen am Port 1 angeschlossenen Joystick abfragt. Sobald Sie das Programm starten, gibt der Amiga bei jeder Joystickbewegung die Richtung aus. Der Druck auf den Feuerknopf beendet das Programm.

```
/* GetJoyData.c
  hardwaremäßige Joystick-Abfrage in C
  Aufruf mit Aztec V3.6:
     cc GetJoyData.c +l
     ln GetJoyData.o -lc32
     GetJoyData
*/

#include <\<>hardware/custom.h>
#include <\<>hardware/cia.h>

void main()
{puts ("Joystick in Port 1 !");
 do /* Hardware-Register abfragen,
       bis Knopf gedrückt wird */

 {if (custom.joy1dat&2) puts("rechts");
  if (custom.joy1dat&512) puts("links");

  if (((custom.joy1dat&1)==1)^((custom.
        joy1dat&2)==2)) puts("unten");

  if (((custom.joy1dat&256)==256)^((custom.
        joy1dat&512)==512)) puts("oben");

 } while (ciaa.ciapra&CIAF_GAMEPORT1);
}                 © 1993 M&T
```

»GetJoyData.c.«: Joystickabfrage in C über die Hardware

Zwecks besserer Lesbarkeit des Quell-Codes haben wir Makro-Definitionen aus den Header-Dateien »custom.h« sowie »cia.h« benutzt.

Möchten Sie nicht Port 1, sondern Port 0 abfragen, ersetzen Sie »joy1dat« durch »joy0dat« sowie »CIAF_GAMEPORT1« durch »CIAF_GAMEPORT0«. *Christof Brühann*

AMOS und Speicher

Viele AMOS-User werden sich schon gefragt haben, wie sie den verfügbaren Speicherplatz abfragen? Da gibt´s zwar den Befehl FREE, aber der zeigt nur den freien Speicher für Variablen. Das nachstehende AMOS-Programm demonstriert, wie man Chip- und Fast-RAM abfragt. *Reik Winkelmann/Faulenrost*

```
Paper 0 : Cls
Print "Chip: ";Chip Free;" Byte"
Print "Fast: ";Fast Free;" Byte"
Print "Chip+Fast : ";Chip Free+Fast Free;
        " Byte"
```

STR und Zahlen

Mit dem BASIC-Befehl x$=STR$(x) wird der numerische Wert von x als eine Zeichenkette nach x$ übergeben. Auch das Vorzeichen. Was bei einer positiven Zahl zu einen Leerstelle führt. Will man mehrere Zeichenketten miteinander verknüpfen. z.B.

```
PRINT x$+y$+z$
```

könnte das Ergebnis so aussehen

```
 12 47 0 58 6 222 1
```

Manchmal sind die Leerzeichen unerwünscht. Folgende Zeile hilft:

```
x$=RIGHT$(STR$(x),LEN(STR$(x))-1)
```

Herbert Pittermann

Was gibt´s für GrafikModi?

Das unten abgedruckte, kurze C-Programm »Grafik_Modus.c« gibt unter Kickstart 2.0 die DisplayID's und Namen der Grafikmodi aus. Eine Ausgabe erfolgt allerdings nur, wenn in der Datei »Mode_Names« ein Name für die entsprechende ID eingetragen ist (diese Datei wird für »BindMonitor« benötigt). Zusätzlich gibt »Grafik_Modus.c« aus, ob der Modus auf dem Rechner verfügbar ist (0 = verfügbar). *Rüdiger Dreier*

```
/* Kompilieren mit -b -v -r -O -Lv */
#include <exec/types.h>
#include <graphics/displayinfo.h>
#include <proto/exec.h>
#include <proto/graphics.h>
#include <proto/dos.h>

 struct GfxBase    *GfxBase;
 struct NameInfo    NI;
 struct DisplayInfo DI;
 ULONG  ID,nextID;

 /* Dieses Programm zeigt die verfügbaren Grafikmodi mit */
 /* Namen an. Es wird ein Name ausgegeben, wenn in       */
 /* Mode_Names ein Name für die ID vorhanden ist.        */

 main()
  {LONG i;
         /* Die Library öffnen */
  GfxBase = OpenLibrary("graphics.library",37);
  if(!GfxBase) return;
  printf("  ID_Dez   ID_Hex              Name  Verfügbar\n");
  do
   {nextID = NextDisplayInfo(ID);    /* nächste ID holen  */

    /* Information über die Verfügbarkeit des */
    /* Darstellungsmodus besorgen             */
    i = GetDisplayInfoData(0,(UBYTE *)&DI,
                          sizeof(DI),
                          DTAG_DISP,
                          ID);
    if(i)
  {  /* Nur wenn vorherige Anforderung geklappt hat...     */
   i = GetDisplayInfoData(0,     /* Namensdaten übertragen */
                          (UBYTE *)&NI,
                          sizeof(NI),
                          DTAG_NAME,
                          ID);
 /* Gibt es für diesen Modus einen Namen? Dann ausgeben    */
  if(i) printf("%8ld %#8lx >%30s<          %hd\n",
             ID,
             ID,
             NI.Name,
             DI.NotAvailable);
     }
   ID = nextID;              /* Zur nächsten ID gehen */
    }
   while(nextID != INVALID_ID);
   CloseLibrary(GfxBase);
   Delay(250);  /* Um beim Start von Workbench was zu sehen */
  }             © 1993 M&T
```

»Grafik_Modus.c«: Die Zahl der Grafikmodi der neuen Amigas ist groß; hiermit behalten Sie den Überblick

Menüs: Alles Einstellungssache

Bei vielen Programmen ist es möglich, bestimmte Einstellungen im Menü vorzunehmen. Die angewählten Menüpunkte werden dann mit dem bekannten Amiga-Haken gekennzeichnet. Hat man viele dieser Optionen anzuwählen, wird die Arbeit bald lästig: rechte Maustaste drücken, in die Titelleiste fahren, Menüpunkt anwählen, Maustaste loslassen, Maustaste wieder drücken, wieder in die Titelleiste fahren, ...

Letzteres kann man sich etwas erleichtern: rechte Maustaste drücken, in die Titelleiste fahren und, während die rechte Maustaste gedrückt bleibt, die entsprechenden Menüpunkte mit der linken Maustaste anwählen. Das erspart viel Arbeit, vor allem, wenn sich mehrere Optionen im selben Menü befinden.

Sean Durkin

Laufwerksimulation

Manchmal sprechen Programme bestimmte Laufwerke direkt an, um z.B. Voreinstellungen zu lesen. Ist aber das betreffende Laufwerk nicht vorhanden, läuft das Program nicht. Abhilfe schafft »DevRen« von der Fish- Disk 378. Mit dem Programm ist es z.B. möglich, das externe Laufwerk »DF2:« beim Amiga 2000 in »DF1:« umzubenennen, was manchmal sehr nützlich sein kann. *Sean Durkin*

Guter Start mit MEmacs

Wer viel mit dem Texteditor »MicroEmacs« von der Extras-Diskette arbeitet, kennt sicher das Problem: Man hat sich z.B. die Funktionstasten mit oft benötigten Floskeln belegt, muß sie aber bei jedem Programmstart wieder neu eingeben. Das ist zeit- und nervenintensiv.

Es gibt jetzt (ab OS2.0) die Möglichkeit, diese Voreinstellungen in einer Datei namens »s:Emacs_pro« zu speichern. Nach dieser Datei sucht MEmacs bei jedem Programmstart. Man gibt dies folgendermaßen ein: Einfach den Befehl so hinschreiben, wie er im Menü von MEmacs steht, z.B.:

```
Read-file "Datei1.txt"
Set mode wrap
Set-Key F1 "AMIGA-Magazin"
```

Die Datei macht folgendes: Zuerst wird das File »Datei1.txt« geladen, danach wird der Modus auf Wordwrap gestellt und die <F1>-Taste mit dem String »AMIGA-Magazin« belegt, d.h., jedesmal, wenn man <F1> drückt, wird der Text »AMIGA-Magazin« geschrieben. Diese »Emacs_pro«-Datei muß im logischen Verzeichnis »s:« stehen. *Ulrich Priesner*

Zeitfragen

Bekanntlich wird mit dem CLI-Kommando

```
setclock opt load
```

die Uhr gestellt. Wer einmal aufpaßt, wird bemerken, daß die Uhr nach einer gewissen Zeit falsch geht. Ein erneutes

```
setclock opt load
```

schafft Abhilfe.

Eine kleine Batch-Datei kann alles automatisch für Sie erledigen:

```
setclock opt load
wait 30 min
execute setuhr
```

Speichern Sie das Ganze dann noch unter »SetUhr« und rufen Sie es auf mit

```
execute setuhr
```

Wenn Sie die Befehle resident laden, geht das Ganze noch schneller und das Laufwerk bleibt ruhig. Unter WB 1.3 kann »opt« weggelassen werden. *Herbert Pittermann*

Versionskontrolle

Haben Sie schon mal versucht, im CLI mit VERSION die Versionsnummer einer Bibliothek (Libraries; meist im Ordner »libs:«) herauszufinden? Z.B. mit dem Befehl

```
Version ......library
```

Alles schön und gut, aber wenn man versucht, ein Laufwerk oder eine Diskette anzusprechen, wird immer nur die Version der Startdiskette angezeigt.

Hier hilft der TYPE-Befehl mit z.B.

```
type df1:libs/arp.library opt h
```

Der Befehl gibt den Inhalt der Bibliothek in hexadezimaler Schreibweise aus. Wenn ca. die ersten 20 Zeilen aufgelistet sind, können Sie das Ganze mit einer beliebigen Taste unterbrechen. Am besten mit <Space> (Leertaste). In der rechten Spalte können Sie dann die Versionsnummer und das Datum herauslesen.

Dafür ist die Shell hervorragend geeignet, weil mit <Curser_hoch>-Taste der letzte Befehl zurückgeholt wird. So kann man eine Diskette nach der anderen einlegen und die neueste Version einer Library erforschen. TYPE kann mit <Ctrl c> abgebrochen werden.

Die ersten paar Zeilen der »arp.library« beispielsweise so aus

```
0000: 000003F3 00000000 00000003 00000000 ................
0010: 00000002 00001091 00000000 00000001 ................
0020: 000003E9 00001091 70274E75 4AFC0000 ........p'NuJ...
0030: 00040000 001E8027 09000000 004D0000 .......'.....M..
0040: 005A0000 00C00000 12415250 20536865 .Z.......ARP She
0050: 6C6C2050 726F6365 73730000 00000000 ll Process......
0060: 12415250 20426163 6B67726F 756E6420 .ARP Background
0070: 5368656C 6C617270 2E6C6962 72617279 Shellarp.library
0080: 00004172 704C6962 2033392E 31202863 ..ArpLib 39.1 (c
0090: 64682F73 64622034 2F392F38 39290A00 dh/sdb 4/9/89)..
00A0: 646F732E 6C696272 61727900 696E7475 dos.library.intu
00B0: 6974696F 6E2E6C69 62726172 79006772 ition.library.gr
00C0: 61706869 63732E6C 69627261 72790065 aphics.library.e
```

Herbert Pittermann

Parameter in Batch-Dateien:

Durch Batch-Dateien ist es möglich, daß durch die Eingabe eines einzigen Kommandos gleich mehrere Befehle ausgeführt werden. Wenn z.B. vor dem Start eines Programms zunächst noch ASSIGN-Anweisungen ausgeführt werden müssen, können diese zusammen mit dem Programmaufruf in eine Batch-Datei geschreiben werden, so daß zum Start nur noch die Batch-Datei aufgerufen werden muß.

Oft kommt es auch vor, daß Parameter übergeben werden müssen. Soll z.B. mit einer Batch-Datei ein beliebiger Quellcode kompiliert und anschließend daß Programm noch aufgerufen werden, so ist eine Batch-Datei mit Parameterübergabe notwendig. Bei diesem Beispiel könnte die Batch-Datei für den C-Compiler Dice folgendermaßen aussehen:

```
.KEY name
dcc <name>.c -o <name>
<name>
```

Mittels

```
.KEY name"
```

wird zunächst »name« als Parameter definiert. Es wird nun im weiteren Verlauf der Batch-Datei <name> jeweils durch den übergebenen Parameter ersetzt. Angenommen, die Batch-Datei heißt »MakeDice«, würde durch »MakeDice test« folgende Befehlssequenz ausgeführt werden:

```
dcc test.c -o test
test
```

Christof Brühann

Suchpattern wie unter DOS: * und #?

AmigaDOS hat unbestritten Vorteile gegenüber MS-DOS. Aber einiges ist unter MS-DOS sicher eleganter gelöst, was man aber beim Amiga nicht missen muß:

Es geht um die sog. Suchpattern: Platzhalter für Zeichen in Zeichenkette, die nicht genau definiert sind oder unbekannt, z.B. wenn man nach Dateien suchen möchte. Bei AmigaDOS ist dies standardmäßig die Zeichenkombination »#?«, bei MS-DOS ein simpler »*«. Wenn man viel mit beiden Systemen arbeitet, ist das doch sehr verwirrend.

Das muß nicht sein: Unter AmigaDOS 2.04 kann relativ einfach auf das alternative Suchpattern – das Zeichen »*« – umgeschaltet werden. In der »DosLibrary«-Struktur gibt es den Eintrag dl_Root, in dem wiederum der Eintrag rn_Flags enthalten ist. Bit 24 (definiert als RNB_WILDSTAR) entscheidet über das Suchpattern. Ist das Bit gesetzt, wird der * als Pattern verwendet, bei gelöschtem Bit das Standardpattern »#?«. Das Ganze funktioniert wie erwähnt erst ab Kickstart 2.04. *ub*

Ein kurzer Tip zum Schluß

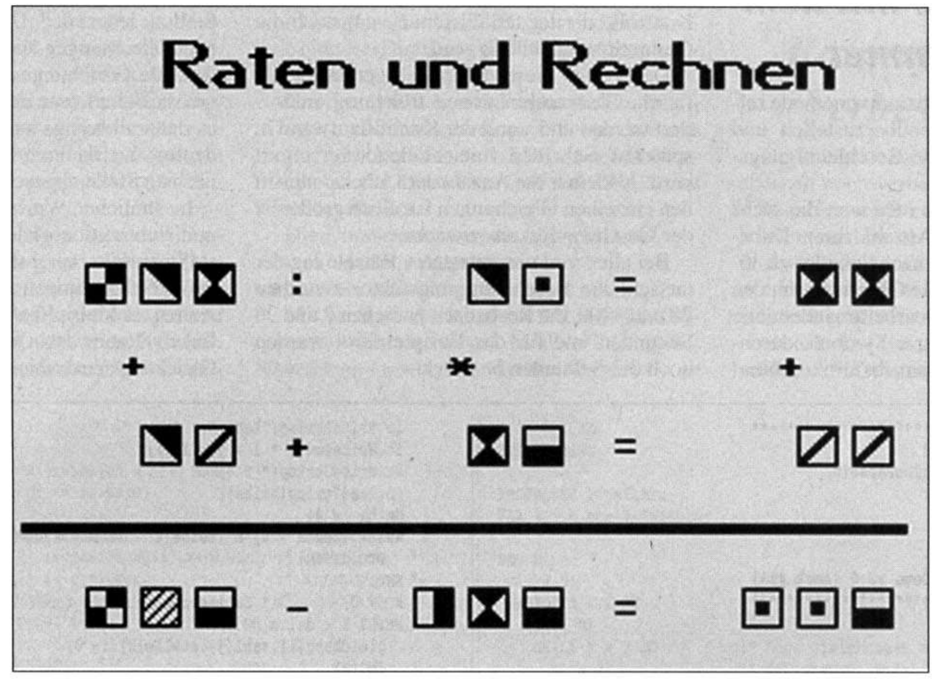
Sie arbeiten doch bestimmt oft mit der Shell oder mit dem CLI und möchten zur Beendigung nicht jedesmal »ENDSHELL« oder »ENDCLI« eingeben.

Ab OS2.0 (Operating System <=> Kickstart 2.0) gibt es die Möglichkeit, mit der Tastenkombination <Ctrl \> (d.h. Sie müssen die Control- und die Backslash-Taste gleichzeitgig drücken) die aktuelle Shell oder das CLI zu verlassen. *ub*

Rätselknacker in Oberon

Der ZahlenGuru

Wie löst man die beliebten Symbolrätsel aus Illustrierten, Rätselheften etc.? Hier ein Lösung in der Programmiersprache Oberon – einer Sprache, die sich gerade auf dem Amiga wachsender Beliebtheit erfreut.



Knifflig: Welches Symbol entspricht welcher Ziffer? Und wie löst man solche Rätsel, wie sie in Illustrierten etc. zu finden sind, mit dem Computer?

von Niels Knoop

ZahlenGuru ist ein Programm zum Lösen von Symbolrätseln, in denen neun verschlüsselte Zahlen im Quadrat zu sechs Gleichungen angeordnet sind. Aufgabe ist es, jedem der maximal zehn Symbole eine Ziffer eindeutig zuzuordnen, so daß die resultierenden Zahlen alle sechs Gleichungen erfüllen. Das folgende Bild zeigt ein Beispiel.

```
       Aufgabe                    Resultat

  abc /  bd =  cc            924 /  21 =  44
    +     *     +              +     *     +
   be +  cf =  ee             26 +  40 =  66
  ----------------           ----------------
  agf - hcf = ddf            950 - 840 = 110

Lösung:
  a    b    c    d    e    f    g    h
  9    2    4    1    6    0    5    8
```

Je nach Symbolzahl gibt es maximal 10! = 3 628 800 Kombinationen, von denen normalerweise nur eine das Gleichungssystem löst. Es gibt aber auch Rätsel, die mehrere Lösungen haben, welche das Programm dann alle finden sollte.

Bedienung:

Das Programm wird am besten vom CLI gestartet, wobei eine Umlenkung der Ein- und Ausgabekanäle wie gewohnt möglich ist, z.B. mit folgendem Kommando:

```
ZahlenGuru <BeispielRaetsel >RAM:Protokoll.
```

ZahlenGuru fragt nach dem Start hintereinander zunächst die (verschlüsselten) Zahlen und dann die Operatoren ab. Deren Reihenfolge ist folgende:

```
Zahl1 Op1 Zahl2  =  Zahl3
 Op4       Op5       Op6
Zahl4 Op2 Zahl5  =  Zahl6

Zahl7 Op3 Zahl8  =  Zahl9
```

Als Symbole dürfen beliebige ASCII-Zeichen verwendet werden mit Ausnahme des Leerzeichens, das für einleitende Leerstellen gedacht ist und dem deshalb immer die Ziffer »0« zugeordnet wird. Führende Leerstellen können, müssen aber nicht mit eingegeben werden, da das Programm die eingegebenen Zahlen automatisch rechtsbündig formatiert.

Jede Zahl darf maximal vier Zeichen lang sein, und da jede Ziffer nur einem Symbol zugeordnet werden kann, dürfen insgesamt nicht mehr als zehn Symbole vorkommen. Als Operatoren werden die Grundrechenarten, symbolisiert durch die Zeichen »+«, »-«, »*« und »/«, akzeptiert.

Über 3 Millionen Kombinationen

Fehleingaben aller Art werden vom Programm gnadenlos ignoriert und nur dadurch kundgetan, daß das Programm keine Lösungen findet. Unsinnige Lösungen sind in einem solchen Fall theoretisch möglich, aber äußerst unwahrscheinlich. Wenn also ein Rätsel mal un-

lösbar erscheint, sollte man immer zuerst prüfen, ob man es richtig eingegeben hat.

Arbeitsweise:

Es ist ziemlich einfach, aber auch zeitintensiv, die Lösungen eines Rätsels durch simples Durchprobieren aller Kombinationen zu finden. Die erste, so arbeitende Programmversion des Autors brauchte zur Lösung von Rätseln mit vollen zehn Symbolen bis zu zwei Stunden, für das zu Beginn stehende Beispiel aus dem AMIGA-Magazin immer noch 24 Minuten. (Alle Zeitmessungen auf einem Amiga 2000).

Die Methode mit dem Holzhammer

Es galt nun, dieser Holzhammermethode raffiniertere Varianten gegenüberzustellen und mit diesen möglichst hohe Beschleunigungsfaktoren zu erzielen.

Die erste, grundlegende Idee war, die sechs Gleichungen nach der Anzahl ihrer Unbekannten zu sortieren und nacheinander zu lösen. Dabei wird als erste die Gleichung mit den wenigsten Unbekannten bearbeitet, indem alle möglichen Belegungen ihrer Symbole durchprobiert werden. Diejenigen Ziffernkombinationen, welche die Gleichung nicht erfüllen, scheiden schon hier aus, während die Lösungen an die nächste Gleichung weitergereicht werden. Da die mit Ziffern belegten Symbole nunmehr als bekannt vorausgesetzt werden dürfen, gelten sie in den noch folgenden Gleichungen nicht mehr als Unbekannte.

Als nächste Gleichung wird aus den übrigen wieder diejenige mit den wenigsten Unbekannten ausgewählt und gelöst, wobei unter Verwendung der Lösungen der letzten Gleichung nur die Belegungen für die "neuen" Symbole durchprobiert werden müssen.

Auf diese Weise werden nun Schritt für Schritt Teilkombinationen ausgesiebt, bis nach Kontrolle der letzten Gleichung nur noch die Gesamtlösungen übrig sind.

Der Vorteil dieser Methode liegt darin, daß falsche Teilkombinationen frühzeitig aussortiert werden und somit der Kontrollaufwand in späteren Schritten entscheidend verringert wird. Je kleiner die Anzahl der Unbekannten in den einzelnen Gleichungen ist, desto größer ist der Geschwindigkeitszuwachs.

Bei allen von uns getesteten Rätseln lag der tatsächliche Beschleunigungsfaktor zwischen 70 und 1200, die Rechenzeit zwischen 3 und 96 Sekunden und für das Beispielrätsel wurden noch 3,3 Sekunden benötigt.

Um das Ganze noch schneller zu bekommen und auch besonders ungünstigen Rätseln beizukommen, fügte der Autor noch einen Programmteil hinzu: das Erweitern der Gleichungsmenge. Die Idee ist dabei, aus den bestehenden Gleichungen neue einstellige Gleichungen herauszuschneiden, die durch ihre geringe Zahl von Unbekannten die Lösung des Rätsels beschleunigen.

Noch schneller mit Trick und ...

Dazu werden zunächst die rechts liegenden Stellen jeder der Ursprungsgleichungen in neue Gleichungen kopiert. Anschließend werden alle Gleichungen wie bisher sortiert und gelöst. Beim Lösen der neuen Gleichungen gilt es dann allerdings zu beachten, daß sie mehrdeutig sind, da ein möglicher Übertrag auf die nächste Stelle abgeschnitten wurde.

In ähnlicher Weise werden bei Additions- und Subtraktionsgleichungen (das Programm stellt zwecks vereinfachter Handhabung ohnehin Subtraktionen zu Additionen und Divisionen zu Multiplikationen um) auch aus dem linken Rand der Originalgleichungen neue Gleichungen extrahiert. Nicht unwichtig für die

```
(**********************************************
Programm  : ZahlenGuru.mod
Zweck     : Lösung von Zahlenrätseln
Version   : 1.3
Autor     : Niels Knoop
Copyright : Public Domain
Sprache   : Oberon
Compiler  : Amiga Oberon Demo v2.0 (Amok #53)
**********************************************)

MODULE ZahlenGuru;

IMPORT io;

TYPE raetselzahl = RECORD
       stelle : ARRAY 4 OF INTEGER
     END;

     raetselgleichung = RECORD
       zahl       : ARRAY 3 OF raetselzahl;
       operator   : CHAR;
       unbekannte : INTEGER;
       enthalten  : ARRAY 11 OF BOOLEAN;
       einstellig,
       vorne      : BOOLEAN
     END;

VAR gleichung : ARRAY 18 OF raetselgleichung;
    symbol : ARRAY 11 OF CHAR;
    ziffer,rangtab : ARRAY 11 OF INTEGER;
    gleichungen,symbole : INTEGER;

PROCEDURE Eingabe;
VAR i,j,k,l,delta,aktsymbole : INTEGER;
    reihe : ARRAY 5 OF CHAR;
    aktzeichen : CHAR;
    speicher : raetselzahl;
    zeichen : CHAR;
BEGIN
  gleichungen := 6;
  symbole := 0;
  symbol[0] := " ";
  ziffer[0] := 0;
  i := 0;
  WHILE i < 3 DO
    j := 0;
    WHILE j < 3 DO
      io.WriteString("Zahl Nr. ");
      io.WriteInt(3 * i + j + 1,1);
      io.WriteString(" ? ");
      io.ReadString(reihe);
      delta := 4;
      WHILE (delta > 0) & (reihe[4 - delta] # CHR(0)) DO
        DEC(delta)
      END;
      k := 0;       (* Zahlen rechtsbündig formatieren *)
      WHILE k < delta DO
        gleichung[i].zahl[j].stelle[k] := 0;
        INC(k)
      END;
      WHILE k < 4 DO
        aktzeichen := reihe[k - delta];
        l := 0;
        WHILE (l < 10) & (l <= symbole) & (symbol[l] # aktzeichen) DO
          INC(l)
        END;
        IF l > symbole THEN          (* Symbole merken und mitzählen *)
          INC(symbole);
          symbol[symbole] := aktzeichen
        END;
        gleichung[i].zahl[j].stelle[k] := l;
        INC(k)
      END;
      INC(j)
    END;
    INC(i)
  END;
  i := 3;
  WHILE i < 6 DO                  (* Vertikale Gleichungen aufstellen *)
    j := 0;
    WHILE j < 3 DO
      gleichung[i].zahl[j] := gleichung[j].zahl[i - 3];
      INC(j)
    END;
    INC(i)
  END;
  i := 0;
  WHILE i < 6 DO
    io.WriteString("Operator Nr. ");
    io.WriteInt(i + 1,1);
```

»ZahlenGuru.mod«: Oberon-Listing zum Knacken von Symbolrätseln (Anfang)

Laufzeit ist dabei, daß nicht nur führende Leerstellen aller drei Zahlen (» « + » « = » «), sondern auch »Nieten« der Form »x« + » « = »x« oder » « + »x« = »x« übersprungen werden, welche, mit einer Unbekannten bevorzugt nach vorne sortiert, von jeder Ziffer erfüllt werden. Auch die »linken« Gleichungen sind mehrdeutig, weil ein möglicher von rechts kommender Übertrag berücksichtigt werden muß.

... Beschleunigung um Faktor 10 000

Weil zum Sortieren nun mehr Gleichungen und solche mit weniger Unbekannten zur Verfügung stehen, wird die Lösung stark beschleunigt, denn je weniger Unbekannte en bloc (also in einer gemeinsamen Gleichung) durchprobiert werden müssen, insbesondere zu Anfang, desto früher werden falsche Lösungen aussortiert – der Rekursionsbaum wird kräftig entlaubt.

Die reine Rechenzeit (ohne Laden und Lösungsausgabe) zur Lösung von einigen Dutzend Rätseln, die wir aus verschiedenen Zeitschriften und Rätselheften zusammengesucht haben und die alle neun oder zehn Symbole hatten, betrug stets weniger als eine Sekunde, im Mittel etwa 0,5 Sekunden; gegebüber der Ur-Version wurden Beschleunigungsfaktoren von 2000 bis über 10000 erzielt.

Zum Programmtext

Das Programm ist in der Programmiersprache Oberon verfaßt (näheres zu Oberon in: »Faszination Programmieren Nr. 1, Seite 67 ff., »Auf Wirth´schen Spuren«, Programmierkurs Oberon). Es wurde mit der Demoversion 2.0 des Amiga-OBERON-Compilers erstellt, die jedem zu empfehlen ist, der sich für diese Programmiersprache interessiert (siehe Seite 114). Sie ist u.a. auf AMOK 53 zu finden, frei kopierbar und enthält eingeschränkte Versionen von Compiler, Linker, Debugger und Editor sowie eine Beschreibung des Oberon-Sprachumfangs und des Compilers.

Um den Quelltext des Programms zu modifizieren, muß man allerdings auf einen anderen Editor zurückgreifen (z.B. ED), da der Oberon-Editor in der Demo-Version nur 200 Zeilen Text verarbeitet.

Die Übersetzungsanweisungen lauten:

```
oberon -dmsvb ZahlenGuru
olink -dms ZahlenGuru
```

Mit wenigen Änderungen läßt sich das Programm auch von einem Modula-2-Compiler übersetzen lassen, da es keinen Gebrauch von Typenerweiterung oder anderen speziellen Oberon-Features macht und damit weitgehend (Modula-2-Fans mögen das Wort verzeihen) »abwärtskompatibel« ist.

Wer sich die Laufzeitverkürzungen veranschaulich will, kann den Aufruf der Prozedur »Erweitern« im Hauptprogramm als Kommentar ausklammern; dann werden nur die ursprünglichen Gleichungen sortiert und bearbeitet. Ganz Hartgesottene können zusätzlich auch den Aufruf von »Sortieren« unterbinden, worauf die Holzhammer-Methode angewandt wird, was dann allerdings wegen anderer Kontrollweise noch etwas länger dauert als in der ursprüngliche Version.

Raum für weitere Verbesserungen

Der vorgestellte Algorithmus kann mit Sicherheit noch verbessert werden – denken Sie z.B. an feinere Sortierkriterien – und auch die Implementation als solche holt wohl nicht die höchstmögliche Geschwindigkeit heraus, da sich der Autor in erster Linie bemüht hat, übersichtlich und verständlich zu bleiben. Versuchen Sie das Ganze doch einmal selbst und schicken Sie uns Ihre Verbesserungen. ■

```
      io.WriteString(" ? ");
      io.ReadString(reihe);
      IF (reihe[0] = "+") OR (reihe[0] = "*") THEN
        gleichung[i].operator := reihe[0]
      ELSE
        speicher := gleichung[i].zahl[0];                  (* Gleichungen *)
        gleichung[i].zahl[0] := gleichung[i].zahl[2];      (* umstellen:  *)
        gleichung[i].zahl[2] := speicher;                  (* x-y=z:z+y=x *)
        IF reihe[0] = "-" THEN                             (* x/y=z:z*y=x *)
          gleichung[i].operator := "+"
        ELSE
          gleichung[i].operator := "*"
        END
      END;

      gleichung[i].einstellig := FALSE;
      gleichung[i].vorne := FALSE;
      gleichung[i].unbekannte := symbole;
      INC(i)
    END;

    i := 1;
    WHILE i <= symbole DO
      rangtab[i] := i;
      INC(i)
    END;
    io.WriteLn
  END Eingabe;

  PROCEDURE Anzeige;
  VAR i : INTEGER;

  BEGIN
    io.WriteString("Lösung:");
    io.WriteLn;
    i := 1;
    WHILE i <= symbole DO
      io.Write(symbol[i]);
      INC(i)
    END;
    io.WriteLn;
    i := 1;
    WHILE i <= symbole DO
      io.WriteInt(ziffer[i],1);
      INC(i)
    END;
    io.WriteLn;
    io.WriteLn
  END Anzeige;

  PROCEDURE Erweitern;
  VAR i,j,k,pos : INTEGER;

  BEGIN
    i := 0;
    WHILE i < 6 DO
      j := 0;
      WHILE j < 3 DO          (* Neue Gleichungen aus dem rechten Rand *)
        k := 0;               (* der Ursprungsgleichungen gewinnen     *)
        WHILE k < 3 DO
          gleichung[gleichungen].zahl[j].stelle[k] := 0;
          INC(k)
        END;
        gleichung[gleichungen].zahl[j].stelle[3] :=
            gleichung[i].zahl[j].stelle[3];
        INC(j)
      END;
      gleichung[gleichungen].operator := gleichung[i].operator;
      gleichung[gleichungen].einstellig := TRUE;
      gleichung[gleichungen].vorne := FALSE;
      INC(gleichungen);
      IF gleichung[i].operator = "+" THEN       (* Bei +/- Gleichungen  *)
        pos:=0;                                 (* auch vom linken Rand *)
        WHILE (pos < 4) &
              (((gleichung[i].zahl[1].stelle[pos] =         (* Nieten *)
                 gleichung[i].zahl[2].stelle[pos]) &        (* x+"=x  *)
                (gleichung[i].zahl[0].stelle[pos] = 0)) OR  (* "+y=y  *)
               ((gleichung[i].zahl[0].stelle[pos] =         (* "+"="  *)
                 gleichung[i].zahl[2].stelle[pos]) &        (* über-  *)
                (gleichung[i].zahl[1].stelle[pos] = 0))) DO (* gehen  *)
          INC(pos)
        END;
        IF (pos < 4) THEN
          j := 0;
          WHILE j < 3 DO
            k := 0;
            WHILE k < 3 DO
              gleichung[gleichungen].zahl[j].stelle[k] := 0;
              INC(k)
            END;
```

»ZahlenGuru.mod«: Oberon-Listing zum Knacken von Symbolrätseln (Fortsetzung)

```
          gleichung[gleichungen].zahl[j].stelle[3] :=
              gleichung[i].zahl[j].stelle[pos];
          INC(j)
        END;
        gleichung[gleichungen].operator := "+";
        gleichung[gleichungen].einstellig := TRUE;
        gleichung[gleichungen].vorne := TRUE;
        INC(gleichungen)
      END
    END;
  INC(i)
  END
END Erweitern;


PROCEDURE Sortieren;
VAR i,j,k,aktsym,min,rangpos : INTEGER;
    speicher : raetselgleichung;
BEGIN
  i := 0;
  WHILE i < gleichungen DO        (* Unbekannte zählen und notieren *)
    gleichung[i].unbekannte := 0;
    j := 1;
    WHILE j < 11 DO
      gleichung[i].enthalten[j] := FALSE;
      INC(j)
    END;
    j := 0;
    WHILE j < 3 DO
      k := 0;
      WHILE k < 4 DO
        aktsym := gleichung[i].zahl[j].stelle[k];
        IF (aktsym > 0) & ~gleichung[i].enthalten[aktsym] THEN
          gleichung[i].enthalten[aktsym] := TRUE;
          INC(gleichung[i].unbekannte)
        END;
        INC(k)
      END;
      INC(j)
    END;
    INC(i)
  END;

  rangpos := 0;
  i := 0;
  WHILE i < gleichungen DO                 (* Gleichungen sortieren *)
    min := i;
    j := i + 1;
    WHILE j < gleichungen DO
      IF gleichung[j].unbekannte < gleichung[min].unbekannte THEN
        min := j
      END;
      INC(j)
    END;
    IF min > i THEN                       (* Gleichung mit wenigsten *)
      speicher := gleichung[i];           (* Unbekannten vorziehen;  *)
      gleichung[i] := gleichung[min];     (* diese in "rangtab" ein- *)
      gleichung[min] := speicher          (* fügen aus den Bestands- *)
    END;                                  (* listen der übrigen Glei-*)
    j := 1;                               (* chungen herausnehmen.   *)
    WHILE j <= symbole DO                 (* Achtung: Hinterher ent- *)
      IF gleichung[i].enthalten[j] THEN   (* hält der Eintrag "unbe- *)
        INC(rangpos);                     (* kannte" der Gleichung i *)
        rangtab[rangpos] := j;            (* Gesamtzahl der Unbek.   *)
        k := i + 1;                       (* in den Gleichungen 0..i *)
        WHILE k < gleichungen DO
          IF gleichung[k].enthalten[j] THEN
            gleichung[k].enthalten[j] := FALSE
          ELSE
            INC(gleichung[k].unbekannte)
          END;
          INC(k)
        END
      END;
      INC(j)
    END;
    INC(i)
  END
END Sortieren;


PROCEDURE Durchlauf(glnum,ebene : INTEGER);
VAR i,j : INTEGER;
    wert0,wert1,wert2 : LONGINT;
    ok : BOOLEAN;
BEGIN
  IF ebene <= gleichung[glnum].unbekannte THEN
    i := 0;
    WHILE i < 10 DO
      ziffer[rangtab[ebene]] := i;     (* Aktuelle Unbek. mit freien *)
      j := 1;                          (* Ziffernwerten 0-9 belegen  *)
      WHILE ziffer[rangtab[j]] # i DO
        INC(j)
      END;
      IF j = ebene THEN
        Durchlauf(glnum,ebene + 1)     (* Weiter zur nächsten Unbek. *)
      END;
      INC(i)
    END
  ELSE                                 (* Gleichung kontrollieren *)
    IF gleichung[glnum].einstellig THEN
      wert0 := ziffer[gleichung[glnum].zahl[0].stelle[3]];
      wert1 := ziffer[gleichung[glnum].zahl[1].stelle[3]];
      wert2 := ziffer[gleichung[glnum].zahl[2].stelle[3]];
      IF gleichung[glnum].vorne THEN
        wert1 := wert0 + wert1;
        ok := (wert1 = wert2) OR (wert1 + 1 = wert2)
      ELSE
        IF gleichung[glnum].operator = "+" THEN
          wert1 := wert0 + wert1;
          ok := (wert1 = wert2) OR (wert1 = wert2 + 10)
        ELSE
          ok := (wert0 * wert1) MOD 10 = wert2
        END
      END
    ELSE

      wert0 := 1000 * ziffer[gleichung[glnum].zahl[0].stelle[0]] +
                100 * ziffer[gleichung[glnum].zahl[0].stelle[1]] +
                 10 * ziffer[gleichung[glnum].zahl[0].stelle[2]] +
                      ziffer[gleichung[glnum].zahl[0].stelle[3]];
      wert1 := 1000 * ziffer[gleichung[glnum].zahl[1].stelle[0]] +
                100 * ziffer[gleichung[glnum].zahl[1].stelle[1]] +
                 10 * ziffer[gleichung[glnum].zahl[1].stelle[2]] +
                      ziffer[gleichung[glnum].zahl[1].stelle[3]];
      wert2 := 1000 * ziffer[gleichung[glnum].zahl[2].stelle[0]] +
                100 * ziffer[gleichung[glnum].zahl[2].stelle[1]] +
                 10 * ziffer[gleichung[glnum].zahl[2].stelle[2]] +
                      ziffer[gleichung[glnum].zahl[2].stelle[3]];
      IF gleichung[glnum].operator = "+" THEN
        ok := wert0 + wert1 = wert2
      ELSE
        ok := wert0 * wert1 = wert2
      END
    END;
    IF ok THEN
      IF glnum < (gleichungen - 1) THEN
        Durchlauf(glnum + 1,ebene)       (* Weiter zur nächsten Gl. *)
      ELSE
        Anzeige
      END
    END
  END
END Durchlauf;

BEGIN
  io.WriteLn;
  io.WriteString("##### ZahlenGuru 1.3 #####");
  io.WriteLn;
  io.WriteLn;
  Eingabe;

  io.WriteString("Meditiere...");
  io.WriteLn;
  io.WriteLn;
  Erweitern;
  Sortieren;
  Durchlauf(0,1);

  io.WriteString("Ende, keine weiteren Lösungen.");
  io.WriteLn;
  io.WriteLn;
  io.closeDelay := 750
END ZahlenGuru.
```


»ZahlenGuru.mod«: Oberon-Listing zum Knacken von Symbolrätseln (Ende)

Schachautomaten

Quevedo geknackt

In der Knobelecke des AMIGA-Magazins 4/93 fragten wir nach einem Programm, das einfache Schachaufgaben lösen soll. Niels Knoop aus Kronberg machte sich an diese komplizierte Programmieraufgabe und schickte uns eine Lösung in Oberon-2.

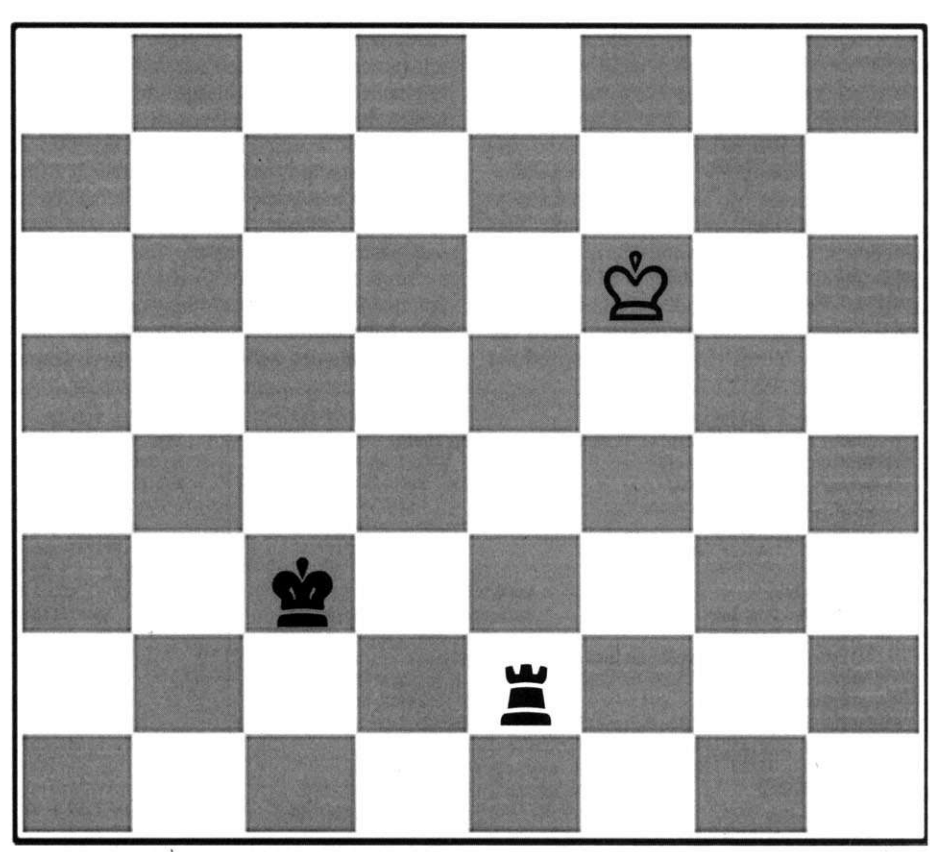
von Niels Knoop

Das Programm Quevedo trägt den Namen des spanischen Erfinders Leonardo Torres y Quevedo, der im Jahre 1890 eine Schachmaschine präsentierte, die das Endspiel mit Turm und König gegen König beherrschte. Verschuldet hat dieses Programm die Knobelecke des AMIGA-Magazins, die in der Ausgabe 4/93 aufrief, auch dem Amiga diese Fähigkeit zu verleihen und dabei die bis zum Matt höchstens erforderliche Zuganzahl von über 60 der Originalmaschine in Richtung der theoretischen Grenze von 16 zu vermindern.

Bedienung:
Quevedo kann von der Shell oder von der Workbench gestartet werden, wobei beim Start von der Shell darauf zu achten ist, daß die Größe des Fensters für die Darstellung des Schachbretts und des Partieverlaufs ausreicht (mindestens 28 Zeilen Text im eingestellten Standard-Font). Wenn das Programm per Icon von der Workbench geladen wird, öffnet es ein eigenes Fenster, dessen Größe man über den TOOLTYPE-Eintrag »WINDOW=...« im Icon einstellen kann. Das Schließen des Fensters nach dem Programmende erfolgt absichtlich erst nach ein paar Sekunden Verzögerung.

Turm und König setzen matt

Nach Eingabe der Ausgangsposition tritt Quevedo mit weißem Turm und König gegen den Benutzer an, der versuchen soll, den schwarzen Turm bestmöglich zu verteidigen. Die Eingabe eines Zugs erfolgt dabei durch Angabe eines Zielfeldes (a4,B8 etc.) oder einer Ziffer (1-4, 6-9), die entsprechend der Anordnung der Tasten im Zahlenblock die Richtung ausdrückt, in die der König ziehen soll. Jede Eingabe muß mit der Return- oder Enter-Taste abgeschlossen werden. Unzulässige Eingaben werden abgefangen und erneut abgefragt. Ein Abbruch des Spiels ist durch Eingabe von <q> möglich. Man sollte etwas vorsichtig mit der Return- und der Enter-Taste umgehen, da die Anzeige der Figuren und des Partieverlaufs sonst etwas durcheinanderkommen kann.



Matt in wieviel Zügen? Kaum zu glauben, aber der Amiga kann sogar solche Aufgaben lösen

Programmtext:
Quevedo wurde in Oberon-2 geschrieben und mit Amiga Oberon 3.0 übersetzt. Wer keinen Oberon-Compiler besitzt, kann auch die Demo-Version 3.0 dieses Compilers verwenden, die unter anderem auf AMOK #75 zu finden ist.

Die wesentlichen Unterprogramme, die Bewertung und Auswahl der Züge vornehmen, wurden samt ihren Hilfsmitteln in das Modul QuevedoRoutinen ausgelagert, während das Hauptmodul Quevedo die Ein- und Ausgabe und den Aufruf der entsprechenden Routinen übernimmt.

Arbeitsweise
Obwohl sich die Situation auf dem Brett mit den gerade mal drei Figuren wesentlich einfacher darstellt als bei einem normalen Schachspiel, verbietet es sich dennoch von vornherein, den jeweils optimalen Zug durch Vorausplanung aller Züge inklusive der schlimmstmöglichen Selbstverteidigung des Gegners bis zum Matt zu ermitteln. Sicherlich kann ein Programm schon wertvolle Informationen bekommen und damit gut spielen, ohne sich gar so weit in die Tiefe zu begeben. Die Frage ist allerdings, ob es überhaupt nötig ist, auf diese Art vorzugehen, oder ob die stark geminderte Komplexität in dieser Art von Endspiel nicht ein einfacheres und schnelleres Vorgehen ermöglicht, welches wohl kaum das Optimum darstellen, diesem aber immerhin nahe kommen kann.

Am Anfang der Arbeit an diesem Programm stand die Entscheidung, einen Weg einzuschlagen, bei dem quasi in Echtzeit nur aus den Informationen der jeweiligen Stellung die unmittelbar möglichen Züge in ihrem Wert eingeschätzt werden und derjenige ausgeführt wird, der dabei am besten abschneidet. Ein Algorithmus, der auf diese Weise funktioniert, »denkt« also nur einen Zug voraus und versteckt seine Taktik in der Bewertung der potentiellen Züge. Die Regeln, die er dabei anwendet, sind, wenn man sie erstmal gefunden hat, zwangläufig recht einfach zu befolgen, was das Vorgehen relativ mechanisch und nachvollziehbar macht – nicht unähnlich also der



Vorstellung vom »Schachautomaten« des Elektroingenieurs Quevedo.

Der Kern des Programms Quevedo besteht folglich in einer Bewertungsprozedur, der jeweils alle durch einen weißen Zug erreichbaren Stellungen unterworfen werden, ungeachtet dessen, wie die Partie bisher verlaufen ist und welche der weißen Figuren ziehen würde. Die Güte der verwendeten Bewertungskriterien entscheidet also darüber, ob und wie gut das Programm seiner Aufgabe gerecht wird.

Um geeignete Maßstäbe zu finden, überlegt man sich am besten zunächst Techniken, wie das Matt herbeigeführt werden kann, und versucht anschließend, die jeweilige Vorgehensweise in die Form von Bewertungskriterien zu gießen. Dabei stellt man fest, daß die verschiedenen Strategien zwangsläufig viele Gemeinsamkeiten haben. Da der schwarze König prinzipiell nur am Rand oder in einer Ecke des Brettes mattgesetzt werden kann, indem der Turm entlang dieses Randes Schach gibt und der weiße König das Ausweichen in Richtung Brettmitte verhindert, wird jede Strategie ein Abdrängen des Gegners an den Rand oder in die Ecke beinhalten. Außerdem darf niemals der Turm aufs Spiel oder der Gegner patt gesetzt werden, was sich in der Bewertung entsprechender Stellungen niederschlagen muß.

Die Methode, mit der Quevedo 2.0 den gegnerischen König matt setzt, besteht im wesentlichen darin, daß der Turm versucht, den schwarzen König gemeinsam mit dem eigenen in einer möglichst kleinen Bretthälfte waagerecht oder senkrecht einzuschließen – wobei die Grenzlinien als eine Art Todesstreifen zu keiner der von ihnen getrennten Hälften gehören – und diese Hälfte kontinuierlich zu verkleinern. Wenn das nicht möglich ist, weil sich einer der Könige an der Grenze befindet, bedrängt der weiße König seinen Widersacher von der Seite, wobei er – bevorzugt orthogonal zur Turmgrenze - auf ihn zugeht und möglichst vermeidet, selbst an die Grenze heranzutreten. Wenn der Turm angegriffen wird oder der Gegner mit einem Tempozug in Zugzwang gebracht werden soll, kann sich der Turm auf dem Grenzstreifen ein sicheres Plätzchen suchen, ohne gewonnenes Terrain zu opfern. Auf diese Weise wird der schwarze König gleichzeitig in zwei Richtungen gedrängt; er muß stets nach einer Seite nachgeben, bis er sich schließlich in der Ecke wiederfindet und nur noch ein Ausweichfeld hat. Aus dieser Lage heraus kann er mit Sonderbehandlung in drei Zügen mattgesetzt werden.

Diese Strategie wird beim Ausprobieren des Programms schnell verständlich, weshalb sie hier nicht weiter ausgeführt zu werden braucht. Sie fand ihren Niederschlag in einer Reihe von Bewertungskriterien. Diese werden für die jeweilige Stellung berechnet, in eine Zahlenform gebracht, in der kleinere Werte für günstigere Eigenschaften stehen, und nach Prioritäten geordnet nebeneinander in der Gesamtwertung versammelt, indem sie nacheinander in die un-

```
(* Programm  : Quevedo.mod
   Zweck     : Schachendspiel Turm und König gegen König
   Version   : 2.0 / 5-8-93
   Autor     : Niels Knoop
   Copyright : Public Domain
   Sprache   : Oberon-2
   Compiler  : Amiga Oberon 3.0                              *)

MODULE Quevedo;
IMPORT QuevedoRoutinen,io;
VAR zuege : INTEGER;
    altpos,neupos : QuevedoRoutinen.stellung;
    zeichenkette : ARRAY 255 OF CHAR;
    abbruch : BOOLEAN;
PROCEDURE LesePosition(VAR x,y : INTEGER) : BOOLEAN;
(* Richtung (per Zahlenblock) oder Zielfeld einlesen *)
VAR eingabe : ARRAY 100 OF CHAR;
    n : INTEGER;
BEGIN
  io.ReadString(eingabe);
  IF eingabe[1] # CHR(0) THEN
    x := ORD(CAP(eingabe[0])) - 64;
    y := ORD(CAP(eingabe[1])) - 48
  ELSE
    n := ORD(eingabe[0]) - 48;
    IF (n >= 1) & (n <= 9) THEN
      x := x + (n - 1) MOD 3 - 1;
      y := y + (n - 1) DIV 3 - 1
    END
  END;
  RETURN (CAP(eingabe[0]) = "Q")
END LesePosition;
PROCEDURE SchreibePosition(x,y : INTEGER);
(* Feldkoordinaten im Klartext ausgeben *)
BEGIN
  io.Write(CHR(x + 96));
  io.Write(CHR(y + 48))
END SchreibePosition;
PROCEDURE GotoXY(x,y : INTEGER);
(* Cursor im Console-Fenster setzen *)
VAR sequenz : ARRAY 10 OF CHAR;
BEGIN
  sequenz := "\[00;00H";
  sequenz[1] := CHR((y DIV 10) MOD 10 + 48);
  sequenz[2] := CHR(y MOD 10 + 48);
  sequenz[4] := CHR((x DIV 10) MOD 10 + 48);
  sequenz[5] := CHR(x MOD 10 + 48);
  io.WriteString(sequenz);
END GotoXY;
PROCEDURE Brettmalen;
(* Schachbrett auf den Bildschirm zeichnen *)
VAR i,j,k : INTEGER;
    schwarz,weiss,leer : ARRAY 5 OF CHAR;
BEGIN
  schwarz := "\[41m";
  weiss := "\[42m";
  leer := "\[40m";
  io.WriteString("\[0m\f \r");  (* sic! *)
  FOR i := 8 TO 1 BY -1 DO
    FOR j := 1 TO 3 DO
      GotoXY(1, 25 - 3 * i + j);
      io.WriteString("   ");
      FOR k := 1 TO 8 DO
        IF (i MOD 2 = k MOD 2) THEN
          io.WriteString(schwarz);
        ELSE
          io.WriteString(weiss);
        END;
        io.WriteString("     ")
      END;
      io.WriteString(leer);
    END;
  END;
  FOR i := 8 TO 1 BY -1 DO
    GotoXY(2, 27 - 3 * i);
    io.Write(CHR(i + 48))
  END;
  FOR i := 1 TO 8 DO
    GotoXY(5 * i + 1, 27);
    io.Write(CHR(i + 64))
  END
END Brettmalen;
PROCEDURE BeschreibeFeld(x,y,farbe : INTEGER; zeichen : CHAR);
(* Angegebenes Feld auf dem Bildschirm beschreiben oder löschen *)
VAR sequenz : ARRAY 25 OF CHAR;
BEGIN
  sequenz := "\[30m\[43m \[1m \[22m \[0m";
  IF (farbe >= 0) THEN
    sequenz[2] := CHR(farbe MOD 4 + 48);
    sequenz[6] := "3";
    sequenz[12]:= zeichen
  ELSE
    sequenz[6] := CHR((x + y) MOD 2 + 49)
  END;
  GotoXY(5 * x, 27 - 3 * y);
  io.WriteString(sequenz)
END BeschreibeFeld;
BEGIN
io.Clear;
io.WriteString("  QUEVEDO 2.0 - Schachendspiel\n");
io.WriteString("   Turm und König gegen König\n");
io.WriteString("(benötigt ca. 80x30 Zeichen Platz)\n");
io.WriteString("==================================\n");
io.WriteString("Bitte die Anfangsstellung eingeben\n");
io.WriteString("(z.B. Weiß: Th8 Ka1, Schwarz: Kd5)\n\n");
LOOP
  io.WriteString("Weiß   : T");
  abbruch := LesePosition(altpos.wtx,altpos.wty);
  io.WriteString("         K");
  abbruch := LesePosition(altpos.wkx,altpos.wky);
```

teren Stellen dieser Zahl geschrieben werden, nachdem dort zuvor durch Stellenverschiebung Platz geschaffen wurde. Das ermöglicht nicht nur einen Vergleich zweier Stellungen mit einem Befehl, sondern auch das Hinzufügen, Herausnehmen und Vertauschen von Kriterien mit minimalem Aufwand.

In eine Art Spielanleitung zur Bewertung umformuliert, lautet die Strategie nun (etwas vereinfacht) folgendermaßen :

Bevorzuge von bis dato gleichwertigen Zügen diejenigen, nach denen

a. der Gegner nicht pattgesetzt ist,
b. der Turm nicht geschlagen werden kann,
c. das Matt erzielt ist,
d. ein Matt im nächsten Zug garantiert ist,
e. ein Matt in zwei Zügen garantiert ist,
f. die Breite der Bretthälfte der Könige am kleinsten ist,
g. die beiden Könige sich bestimmten Maßen zufolge am nächsten sind,
h. der Turm am sichersten ist.

Die obere Anforderung an das Programm ist natürlich, daß es wirklich aus jeder beliebigen Ausgangsstellung mattsetzen kann. Vereitelt werden könnte dies noch durch ein Remis, das durch eine zyklische Wiederholung von Stellungen insbesondere durch direktes Hin- und Herziehen herbeigeführt werden kann, was eine schlimme Falle für den Algorithmus bedeutet. Denn formal zu beweisen, daß ein Verfahren niemals Zyklen produziert, ist eine ungemein aufwendige Angelegenheit.

Das Prinzip, aus jeder Stellung heraus immer den gleichen Zug auszuführen, ungeachtet dessen, wann sie in der Partie erreicht wird, hilft aus dieser Verlegenheit heraus, da es ermöglicht, mit einer modifizierten Programmversion in wenigen Minuten (wozu hat man schließlich eine Turbokarte...) das Bewertungsverfahren auf Zyklen zu überprüfen. Gegenüber den gigantischen Zugkombinationen, die möglich sind, gibt es ja nur eine vergleichsweise überschaubare Anzahl legaler Stellungen der drei Figuren. Sie zählt 175168 Positionen (wenn Weiß am Zug ist), und wenn man einmal herausgefunden hat, daß eine Stellung mit dem vorgegebenen Algorithmus in n Zügen spätestens zum Matt führt, kann man dieses Wissen speichern und für andere Spielverläufe verwenden, die zu dieser Stellung führen, was die Rechenzeit erheblich verkürzt.

Das Ergebnis verheißt, daß Quevedo zyklenfrei ist; der Worst Case beträgt 26 Züge. Allerdings dürfte das Auffinden einer entsprechenden Ausgangsposition, ebenso wie die optimale Verteidigung ohne genaues Studium des Programms schwierig sein – und von der im AMIGA-Magazin als besonders ungünstig präsentierten Stellung wTh8, wKa1, sKd5 aus werden nie mehr als 20 Züge benötigt.

Copyright und ein kleiner Wink:

Quevedo ist ein Public-Domain-Programm, das frei benutzt, kopiert und weiterentwickelt werden darf – viel Spaß ! ♜

```
    io.WriteString("Schwarz: K");
    abbruch := LesePosition(altpos.skx,altpos.sky);
    IF QuevedoRoutinen.LegaleStellung(altpos) THEN
      EXIT
    END;
    io.WriteString("\nSorry, illegale Stellung !\n\n")
  END;
  io.WriteLn;
  Brettmalen;
  BeschreibeFeld(altpos.wtx,altpos.wty,2,"T");
  BeschreibeFeld(altpos.wkx,altpos.wky,2,"K");
  BeschreibeFeld(altpos.skx,altpos.sky,1,"K");
  abbruch := FALSE;
  zuege := 1;
  neupos := altpos;
  REPEAT
    QuevedoRoutinen.WeisserZug(neupos);
    GotoXY(49,zuege + 1);
    io.WriteInt(zuege,2);
    io.WriteString(".  ");
    IF neupos.turmzug THEN
      io.Write("T");
      SchreibePosition(altpos.wtx,altpos.wty);
      io.Write("-");
      SchreibePosition(neupos.wtx,neupos.wty);
      BeschreibeFeld(altpos.wtx,altpos.wty,-1," ");
      BeschreibeFeld(neupos.wtx,neupos.wty,2,"T")
    ELSE
      io.Write("K");
      SchreibePosition(altpos.wkx,altpos.wky);
      io.Write("-");
      SchreibePosition(neupos.wkx,neupos.wky);
      BeschreibeFeld(altpos.wkx,altpos.wky,-1," ");
      BeschreibeFeld(neupos.wkx,neupos.wky,2,"K")
    END;
    IF neupos.schach THEN
      GotoXY(60,zuege + 1);
      io.Write("+")
    END;
    IF ~neupos.matt THEN
      GotoXY(62,zuege + 1);
      io.Write("K");
      SchreibePosition(altpos.skx,altpos.sky);
      io.Write("-");
      REPEAT
        neupos.skx := altpos.skx;
        neupos.sky := altpos.sky;
        GotoXY(66,zuege + 1);
        abbruch := LesePosition(neupos.skx,neupos.sky);
        GotoXY(66,zuege + 1);
        io.WriteString("\[K")
      UNTIL abbruch OR QuevedoRoutinen.LegaleStellung(neupos) &
            ((neupos.skx # altpos.skx) OR (neupos.sky # altpos.sky)) &
            (ABS(neupos.skx - altpos.skx) <= 1) &
            (ABS(neupos.sky - altpos.sky) <= 1);
      IF ~abbruch THEN
        BeschreibeFeld(altpos.skx,altpos.sky,-1," ");
        BeschreibeFeld(neupos.skx,neupos.sky,1,"K");
        GotoXY(66,zuege + 1);
        SchreibePosition(neupos.skx,neupos.sky)
      END
    ELSE
      io.Write("+")
    END;
    altpos := neupos;
    INC(zuege)
  UNTIL neupos.matt OR abbruch;
  GotoXY(1,28);
  io.closeDelay := 250
END Quevedo.
```


»Quevedo.mod«: Oberon-2-Programm, um Turmendspiele zu lösen (wichtige Funktionen siehe Listing unten)

```
(* QuevedoRoutinen.mod Version 2.0 / 5-8-93, Kern von Quevedo *)

MODULE QuevedoRoutinen;

TYPE stellung* = RECORD
                   wtx*,wty*,wkx*,wky*,skx*,sky* : INTEGER;
                   schach*,matt*,turmzug* : BOOLEAN;
                   bewertung : LONGINT
                 END;

PROCEDURE Mini(a,b : INTEGER) : INTEGER;
BEGIN
  IF (a < b) THEN
    RETURN a
  ELSE
    RETURN b
  END
END Mini;

PROCEDURE Maxi(a,b : INTEGER) : INTEGER;
BEGIN
  IF (a > b) THEN
    RETURN a
  ELSE
    RETURN b
  END
END Maxi;
```

»QuevedoRoutinen.mod«: Kernroutinen des Schachprogramms als Extra-Modul (Anfang)



```
PROCEDURE Zahl(ausdruck : BOOLEAN) : INTEGER;
(* FALSE -> 0, TRUE -> 1 *)
BEGIN
  IF (ausdruck) THEN
    RETURN 1
  ELSE
    RETURN 0
  END
END Zahl;
PROCEDURE Bedroht(s : stellung; x,y : INTEGER) : BOOLEAN;
(* Bedroht der weiße Turm oder König in Stellung s das Feld (x,y) ? *)
BEGIN
 RETURN (Maxi(ABS(s.wkx - x),ABS(s.wky - y)) = 1) OR
        (s.wtx = x) & ((s.wkx # x) OR ((s.wky < s.wty)=(s.wky < y))) OR
        (s.wty = y) & ((s.wky # y) OR ((s.wkx < s.wtx)=(s.wkx < x)))
END Bedroht;
PROCEDURE LegaleStellung*(s : stellung) : BOOLEAN;
(* Ist die Stellung s legal, wenn weiß am Zug ist ? *)
BEGIN
  RETURN (s.wtx >= 1) & (s.wtx <= 8) & (s.wty >= 1) & (s.wty <= 8) &
         (s.wkx >= 1) & (s.wkx <= 8) & (s.wky >= 1) & (s.wky <= 8) &
         (s.skx >= 1) & (s.skx <= 8) & (s.sky >= 1) & (s.sky <= 8) &
         ((s.wtx # s.wkx) OR (s.wty # s.wky)) &
         ((s.wkx # s.skx) OR (s.wky # s.sky)) &
         ((s.skx # s.wtx) OR (s.sky # s.wty)) &
         ~Bedroht(s,s.skx,s.sky)
END LegaleStellung;
PROCEDURE BewerteStellung(VAR s : stellung);
(* Stellung s bewerten und ggf. die Variablen schach und matt setzen *)
VAR zeilen,spalten,reihen,deltawksk,deltawksk2,deltawtwk,deltawtsk,
    distanzx,distanzy,distanz,turmgefahr1,turmgefahr2,x,y : INTEGER;
    baldmatt,fastmatt,patt,turmbedroht : BOOLEAN;
BEGIN
  s.schach := Bedroht(s,s.skx,s.sky);
  s.matt := FALSE;
  patt := TRUE;
  FOR x := Maxi(s.skx - 1,1) TO Mini(s.skx + 1,8) DO
    FOR y := Maxi(s.sky - 1,1) TO Mini(s.sky + 1,8) DO
      patt := patt & (Bedroht(s,x,y) OR (x = s.skx) & (y = s.sky))
    END
  END;
  IF patt & s.schach THEN
    patt := FALSE;
    s.matt := TRUE
  END;
  deltawtwk := Maxi(ABS(s.wkx - s.wtx),ABS(s.wky - s.wty));
  deltawtsk := Maxi(ABS(s.skx - s.wtx),ABS(s.sky - s.wty));
  deltawksk := Maxi(ABS(s.skx - s.wkx),ABS(s.sky - s.wky));
  deltawksk2:= ABS(s.skx - s.wkx) + ABS(s.sky - s.wky);
  turmgefahr1 := deltawtwk - deltawtsk + 8;
  turmgefahr2 := 8 - deltawtsk;
  turmbedroht := (deltawtsk = 1) & (deltawtwk > 1);
  IF (s.wtx < s.skx) & (s.wtx < s.wkx) THEN
    spalten := 8 - s.wtx
  ELSIF (s.wtx > s.skx) & (s.wtx > s.wkx) THEN
    spalten := s.wtx - 1
  ELSE
    spalten := 9
  END;
  IF (s.wty < s.sky) & (s.wty < s.wky) THEN
    zeilen := 8 - s.wty
  ELSIF (s.wty > s.sky) & (s.wty > s.wky) THEN
    zeilen := s.wty - 1
  ELSE
    zeilen := 9
  END;
  distanzx := ABS(s.wkx - s.skx +
                 (s.wtx - s.skx) * Zahl(ABS(s.wtx - s.skx) <= 1));
  distanzy := ABS(s.wky - s.sky +
                 (s.wty - s.sky) * Zahl(ABS(s.wty - s.sky) <= 1));
  IF (spalten < zeilen) OR (spalten=zeilen) & (distanzx<distanzy) THEN
    reihen := spalten;
    distanz := distanzx
  ELSE
    reihen := zeilen;
    distanz := distanzy
  END;
  baldmatt := (Mini(s.skx,9 - s.skx) = 1) & (Mini(s.sky,9 - s.sky)=1) &
              (deltawksk = 2) & (deltawksk2 = 3);
  fastmatt := (deltawksk = 2) &
              ((Mini(s.skx,9 - s.skx) = 1) &
               ((Mini(s.sky,9 - s.sky) = 2) & (zeilen = 2) OR
                (ABS(s.wty - s.sky) = 1) & (ABS(s.wty - s.wky) = 2)) OR
               (Mini(s.sky,9 - s.sky) = 1) &
               ((Mini(s.skx,9 - s.skx) = 2) & (spalten = 2) OR
                (ABS(s.wtx - s.skx) = 1) & (ABS(s.wtx - s.wkx) = 2)));
  s.bewertung := 0;
  s.bewertung := s.bewertung * 2 + Zahl(patt);
  s.bewertung := s.bewertung * 2 + Zahl(turmbedroht);
  s.bewertung := s.bewertung * 2 + Zahl(~s.matt);
  s.bewertung := s.bewertung * 2 + Zahl(~fastmatt);
  s.bewertung := s.bewertung * 2 + Zahl(~baldmatt);
  s.bewertung := s.bewertung * 8 + reihen;
  s.bewertung := s.bewertung * 8 + distanz;
  s.bewertung := s.bewertung * 8 + deltawksk;
  s.bewertung := s.bewertung * 8 + deltawksk2;
  s.bewertung := s.bewertung * 8 + turmgefahr1;
  s.bewertung := s.bewertung * 8 + turmgefahr2
END BewerteStellung;
PROCEDURE WeisserZug*(VAR s : stellung);
(* Alle für Weiß möglichen Züge ausprobieren, den besten ausführen *)
VAR i,x,y,xmin,xmax,ymin,ymax,anzahl,optimum : INTEGER;
    minimum : LONGINT;
    pos : ARRAY 25 OF stellung;
BEGIN
  anzahl := 0;
  xmin := 1;    xmax := 8;
  ymin := 1;
  ymax := 8;
  IF s.wtx = s.wkx THEN
    IF s.wty < s.wky THEN
      ymax := s.wky - 1
    ELSE
      ymin := s.wky + 1
    END
  ELSIF s.wty = s.wky THEN
    IF s.wtx < s.wkx THEN
      xmax := s.wkx - 1
    ELSE
      xmin := s.wkx + 1
    END
  END;
  FOR x := xmin TO xmax DO
    IF (x # s.wtx) THEN
      pos[anzahl] := s;
      pos[anzahl].wtx := x;
      pos[anzahl].turmzug := TRUE;
      BewerteStellung(pos[anzahl]);
      INC(anzahl)
    END
  END;
  FOR y := ymin TO ymax DO
    IF (y # s.wty) THEN
      pos[anzahl] := s;
      pos[anzahl].wty := y;
      pos[anzahl].turmzug := TRUE;
      BewerteStellung(pos[anzahl]);
      INC(anzahl)
    END
  END;
  FOR x := Maxi(s.wkx - 1,1) TO Mini(s.wkx + 1,8) DO
    FOR y := Maxi(s.wky - 1,1) TO Mini(s.wky + 1,8) DO
      IF ((x # s.wkx) OR (y # s.wky)) &
         ((x # s.wtx) OR (y # s.wty)) &
         (Maxi(ABS(x - s.skx),ABS(y - s.sky)) > 1) THEN
        pos[anzahl] := s;
        pos[anzahl].wkx := x;
        pos[anzahl].wky := y;
        pos[anzahl].turmzug := FALSE;
        BewerteStellung(pos[anzahl]);
        INC(anzahl)
      END
    END
  END;
  optimum := 0;
  minimum := pos[0].bewertung;
  FOR i := 1 TO anzahl - 1 DO
    IF (pos[i].bewertung < minimum) THEN
      optimum := i;
      minimum := pos[i].bewertung
    END
  END;
  s := pos[optimum]
END WeisserZug;
END QuevedoRoutinen.
```


»QuevedoRoutinen.mod«: Kernroutinen des Schachprogramms als Extra-Modul (Ende)





Assembler optimieren

Acht Damen im Schach

Wie optimiert man Assembler-Programme und wie bindet man Assembler-Routinen in C ein? Hierzu ein schönes Beispiel von L.M.M. Veugen, der uns eine Lösung zu einer Knobelaufgabe schickte, bei der es darum ging, ein Programm zu schreiben, das acht Damen auf einem Schachbrett plaziert, ohne daß die Ladies sich gegenseitig bedrohen.

von L.M.M. Veugen

Das C-Programm »DameTest.c« löst die Aufgabe, acht Damen auf einem Schachbrett zu verteilen. Das Programm ermittelt allerdings nur die Zahl der verschiedenen Möglichkeiten, gibt diese allerdings nicht aus. Das zweite Programm »DameTest.OUTPUT« zeigt dazu alle Stellungen.

Das Hauptprogramm ist in der Sprache C geschrieben, die Berechnungsfunktion (»DameCalc«) in Assembler.

Sie starten das Programm wie folgt:

```
DameTest [n]
```

mit

```
n: Anzahl der Berechnungen
```

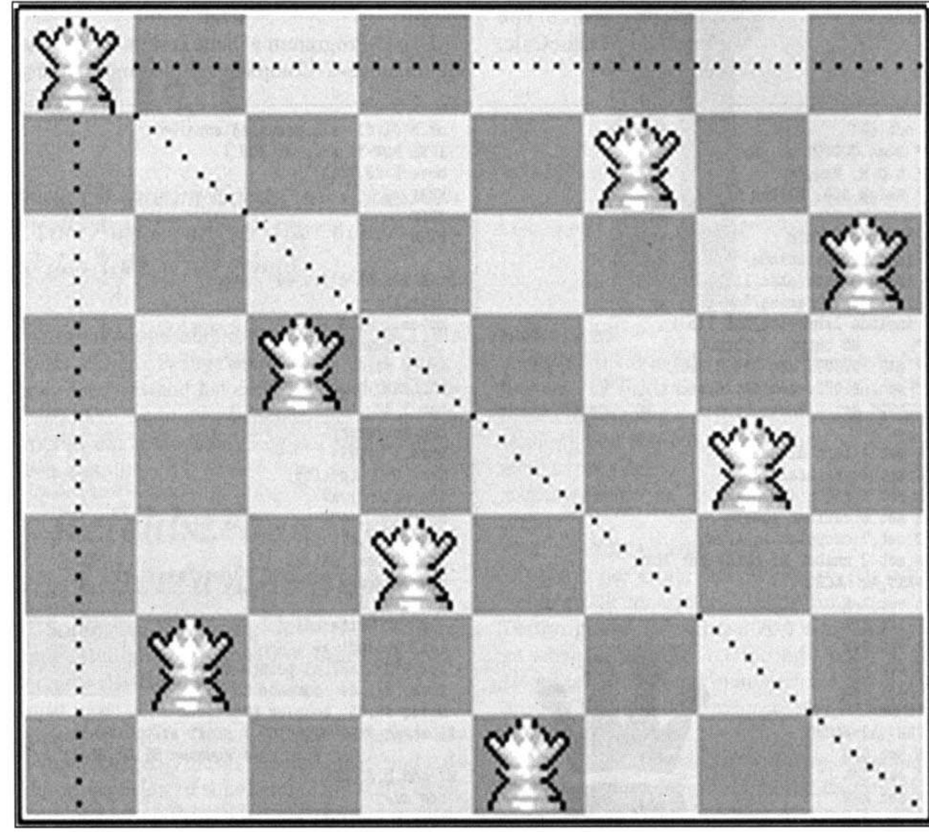
Das Argument »n« ist per default 1. Wenn »n« größer als 1 ist, werden für eine genaue Bestimmung der Zeitdauer der Berechnungsroutine n Berechnungen ausgeführt.

Die Berechnungszeiten (in Sekunden) auf einem Amiga (68000 CPU, 2.3 MByte) sind:
– 0.0754 s (ohne Ausgabe) und
– 39.16 s (mit Ausgabe).

Das Beispiel in Assembler im AMIGA-Magazin 5/92 brachte es auf die Zeiten: 0.3072 s bzw. 39.60 s. Die vorgestellte Berechnungsfunktion ist also ungefähr viermal schneller.

Der verwendete Algorithmus ähnelt dem im AMIGA-Magazin vorgestellten Beispiel. Die Gründe für die höhere Geschwindigkeit sind:
1. Man braucht nur die Hälfte der Stellungen zu untersuchen, denn die Senkrechte durch die Brettmitte ist eine Symmetrielinie.

In Ausgabe 5/92 5/1992, Seite 54 des AMIGA-Magazins wurde ein Programm vorgestellt, welches das Acht-Damen-Problem löst: Es geht dabei darum, acht Damen auf einem Schachbrett aufzubauen, ohne daß eine Dame eine andere bedroht. Das hier vorgestellte Programm bietet eine kombinierte Lösung in C und Assembler, die um einiges schneller ist, als das zuerst vorgestellte Programm.



»Acht Damen«: Eine von 92 Stellungen, in denen sich die Figuren nicht bedrohen; finden Sie die anderen 91 Möglichkeiten? Der Amiga kann´s.

2. Es gibt keine Schleife, sondern die sieben/acht Durchgänge sind mit Makros programmiert.
3. Es wird nur einmal pro neue Reihe eine Maske mit freien Feldern berechnet.
4. Die benutzten Assemblerbefehle sind effizient, z.B. verwendet das Programm keine Adreßregister-indirekte-Befehle.

Auf den als Public-Domain erhältlichen Disketten zu diesem Sonderheft finden Sie in

```
/*********************************
* DameTest.c  by                 *
* L.M.M. Veugen                  *
* Amiga 500, Lattice 5.04        *
*********************************/

#include <stdio.h>
#include <time.h>

void __asm DameCalc( void );

static unsigned int tmBegin[2], tmEnd[2];

static void print_time()
{while ( tmEnd[1] < tmBegin[1] )
    tmEnd[0]--, tmEnd[1] += 1000000;
  printf( "TIME = %d.%06d seconds\n",
   tmEnd[0]-tmBegin[0], tmEnd[1]-tmBegin[1] );
}

int main( argc, argv )
  int argc;
  char *argv[];
{ long  n, n_comps = 0;

  if ( argc > 1 )
    n_comps = atoi( argv[1] );

  timer( tmBegin );
  DameCalc();
  timer( tmEnd );
  printf( "1 computation :\n" );
  print_time();
  Delay(50);

  if ( n_comps )
  { timer( tmBegin );
    for ( n=0; n<n_comps; n++ )
      DameCalc();
    timer( tmEnd );
    printf( "%d computations :\n", n_comps );
    print_time();
  }
}
```


»DameTest.c«: Hauptprogramm, um alle Lösungen zu berechnen



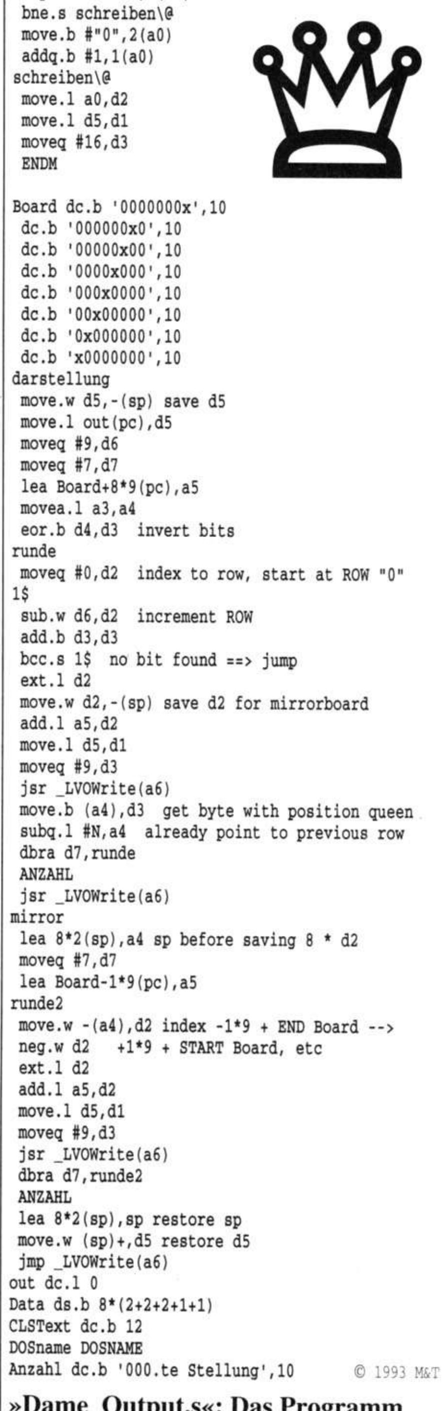


Verzeichnis »df0:DAME« u.a. die folgenden Listings und Programme:

DameTest.c	C-Quelltext des Hauptprogramms
DameTest	lauffähiges Hauptprogramm, ohne Ausgabe
DameTest.OUTPUT	lauffähiges Hauptprogramm, mit Ausgabe
Dame.s	Assembler-Source von Funktion DameCalc, ohne Ausgabe
Dame.OUTPUT.s	Assembler-Source von DameCalc, aber mit Ausgabe (dasselbe wie »Dame.s«, nur Makro OUTPUT gesetzt)
generate	script-Datei, die »DameTest« und »DameTest.OUTPUT« zusammensetzt

Das C-Programm »DameTest.c« wurde mit dem Lattice-C-Compiler V5.04 kompiliert und die Assemblerroutine »Dame.s« mit DevPac V2.14 übersetzt. Die folgende Tabelle zeigt die Übersetzungsanweisungen. ♛

```
; lc DameTest.c
genim2 Dame.s -l
blink FROM LIB:c.o DameTest.o Dame.o TO
      DameTest LIB LIB:lc.lib LIB:amiga.lib
genim2 Dame.OUTPUT.s -l
blink FROM LIB:c.o DameTest.o Dame.OUTPUT.o
      TO DameTest.OUTPUT LIB LIB:lc.lib
      LIB:amiga.lib
```



```
 opt o+
* Dame.OUTPUT.s   by          *
* L.M.M. Veugen               *
* Amiga 500, DevPac 2.14      *

 XDEF _DameCalc
 incdir "DP:include/"
 include exec/exec_lib.i
 include libraries/dos.i
 include libraries/dos_lib.i
*      NO output   output
* M&T  0.3072 sec 39.60 sec
* me   0.0754 sec 39.16 sec
OUTPUT set 0 0 = output / 1 = NO output

L set 0 left diagonal
C set 2 vertical
R set 4 right diagonal
B set 6 current square
T set 7 occupied squares
N set 8 number of bytes per ROW
NEXT_A2 MACRO
L set L+N
C set C+N
R set R+N
B set B+N
T set T+N
 ENDM
PREV_A2 MACRO
L set L-N
C set C-N
R set R-N
B set B-N
T set T-N
 ENDM
NEXTROW MACRO
ROW set \3
 IFGE ROW-5
 cmp.b d4,d3  compare d3 to %11111111
 beq.s back\@  no free squares > jump
 ENDC
 moveq #%00000001,d0 start at square Hx
neuespalte\@
 move.b d0,d1  copy candidate square
 and.b d3,d1  AND with occupied squares
 beq.s neuereihe\@ candidate is free >jump
\1    * R(k)
 add.b d0,d0  square to the left
 bcc.s neuespalte\@ not beyond left border >jump
back\@
 PREV_A2
 move.b B(a2),d0 restore current square
 IFNE ROW-2
 move.b T-N(a2),d3 restore occupied squares
 ENDC
 bra.s \2  * R(k-1)
 NEXT_A2
neuereihe\@
 move.b d0,B(a2)
 movem.w L-N(a2),d1/d2/d3
 or.b d0,d1  occupied squares down-vertical
 or.b d0,d2
 lsr.b #1,d2  occupied squares down-left
 or.b d0,d3
 add.b d3,d3  occupied squares down-right
 IFNE ROW-7  skip at ROW 7
 movem.w d1/d2/d3,L(a2)
 ENDC
 or.b d2,d3
 or.b d1,d3  all occupied squares
 IFNE ROW-7  skip at ROW 7
 move.b d3,T(a2)
 NEXT_A2
 ENDC
 ENDM

REGS REG d1-d7/a0-a6
_DameCalc
 movem.l REGS,-(sp)
 lea DOSname(pc),a1
 moveq #0,d0
 CALLEXEC OpenLibrary
 tst.l d0
 beq nolibrary
 move.l d0,a6
 jsr _LVOOutput(a6)
 lea out(pc),a0
 move.l d0,(a0)
 move.l d0,d1
 lea CLSText(pc),a0
 move.l a0,d2
 moveq #1,d3
 jsr _LVOWrite(a6)
 lea Data(pc),a2
 lea 6*N+B(a2),a3 points to ROW_7
 moveq #-1,d4  compare to d3
 moveq #0,d5  counter positions
S1 moveq #%00001000,d0 ; start at square E1 >
                       ; squares D1,C1,B1,A1
R1 add.b d0,d0
 bcs aus
 move.b d0,B(a2)
 move.b d0,d2
 lsr.b #1,d2  occupied squares down-left
 move.b d0,d3
 add.b d3,d3  occupied squares down-right
 movem.w d0/d2/d3,L(a2)
 or.b d2,d3
 or.b d0,d3  all occupied squares
 move.b d3,T(a2)
 NEXT_A2
S2 NEXTROW R2,R1,2
S3 NEXTROW R3,R2,3
S4 NEXTROW R4,R3,4
S5 NEXTROW R5,R4,5
S6 NEXTROW R6,R5,6
S7 NEXTROW R7,R6,7
S8 cmp.b d4,d3  d3 = %11111111 ==> no 8th queen
 bne.s treffer  unequal ==> jump
 move.b T-N(a2),d3
 bra.s R7  back to ROW 7
treffer
 addq.w #2,d5
 IFEQ OUTPUT
 bsr.s darstellung
 move.b B(a2),d0 output corrupts d0
 ENDC
 move.b T-N(a2),d3
 bra.s R7  back to ROW 7
aus
 move.l a6,a1
 CALLEXEC CloseLibrary
nolibrary
 movem.l (sp)+,REGS
 moveq #0,d0
 rts
ANZAHL MACRO
 lea Anzahl(pc),a0
 addq.b #1,2(a0)
 cmpi.b #"9"+1,2(a0)
 bne.s schreiben\@
 move.b #"0",2(a0)
 addq.b #1,1(a0)
schreiben\@
 move.l a0,d2
 move.l d5,d1
 moveq #16,d3
 ENDM

Board dc.b '0000000x',10
 dc.b '000000x0',10
 dc.b '00000x00',10
 dc.b '0000x000',10
 dc.b '000x0000',10
 dc.b '00x00000',10
 dc.b '0x000000',10
 dc.b 'x0000000',10
darstellung
 move.w d5,-(sp) save d5
 move.l out(pc),d5
 moveq #9,d6
 moveq #7,d7
 lea Board+8*9(pc),a5
 movea.l a3,a4
 eor.b d4,d3  invert bits
runde
 moveq #0,d2  index to row, start at ROW "0"
1$
 sub.w d6,d2  increment ROW
 add.b d3,d3
 bcc.s 1$  no bit found ==> jump
 ext.l d2
 move.w d2,-(sp) save d2 for mirrorboard
 add.l a5,d2
 move.l d5,d1
 moveq #9,d3
 jsr _LVOWrite(a6)
 move.b (a4),d3  get byte with position queen
 subq.l #N,a4  already point to previous row
 dbra d7,runde
 ANZAHL
 jsr _LVOWrite(a6)
mirror
 lea 8*2(sp),a4 sp before saving 8 * d2
 moveq #7,d7
 lea Board-1*9(pc),a5
runde2
 move.w -(a4),d2 index -1*9 + END Board -->
 neg.w d2   +1*9 + START Board, etc
 ext.l d2
 add.l a5,d2
 move.l d5,d1
 moveq #9,d3
 jsr _LVOWrite(a6)
 dbra d7,runde2
 ANZAHL
 lea 8*2(sp),sp restore sp
 move.w (sp)+,d5 restore d5
 jmp _LVOWrite(a6)
out dc.l 0
Data ds.b 8*(2+2+2+1+1)
CLSText dc.b 12
DOSname DOSNAME
Anzahl dc.b '000.te Stellung',10
```


»Dame_Output.s«: Das Programm inklusive Ausgaberoutine



Programmoptimierung

Taktzähler

Das Programm »Taktik.asm« ist die optimale Lösung im Kampf um Taktzyklen und Geschwindigkeitsoptimierung. Es mißt die Ausführungszeit der in der Testschleife eingesetzten Befehle, und dies auf den Taktzyklus genau.

von Gerald Steffens

Man kann sich mit dem unten vorgestellten Programm (»Taktik.asm«) über die Taktzahl eines einzelnen Befehls oder aber einer etwas längeren Befehlsfolge informieren. »Taktik.asm« erfüllt einen ähnlichen Zweck wie eine Taktzyklentabelle. Es lassen sich aber auch einige überraschende Feststellungen machen.

In den normalen Takttabellen steht z.B. bei den Multiplikations- und Divisionsbefehlen, daß es sich um Maximalwerte handelt. Die meisten Programmierer, darunter auch Profis wie J.A.Toebes VIII (siehe AMIGA 3/90), die in Zeitschriften über Programmoptimierung geschrieben haben, nehmen als Taktzahl für Befehle wie

```
MULU #10,D0 , MULU #40,D0
```

oder

```
MULU #640,D0
```

diesen Maximalwert an. In diesem Fall wären das 74 Takte. Anschließend führt man zur Beschleunigung eine Ersatzroutine mit Shift- und Additionsbefehlen vor. Diese hätten im obigen Fall rund 32, 38 bzw. 46 Takte. Nun verkauft man dem staunenden Leser die Differenz von 28 bis 42 Takten als riesige Zeitersparnis. Bei einer genauen Untersuchung mit dem Programm »taktik« erfährt man aber, daß obige Multiplikationsbefehle nur 46 statt 74 Takte verbrauchen. Reingewinn somit 0-14 Takte, und der Aufwand hat sich kaum gelohnt. Insbesondere wenn man bedenkt, daß die Ersatzroutine ein zusätzliches Register zum Rechnen benötigt.

Befehlszeiten: viele Überraschungen

Sollte man z.B. aus irgendwelchen Gründen eine Multiplikation mit 999 benötigen, verbraucht der Befehl

```
MULU #999,D0
```

genau 58 Takte. Ich empfehle es jedem als Übungsaufgabe, dieses mit den Shift- und Additionsbefehlen zu simulieren. Und sollte es einem gewieften Programmierer gelingen, diese Zeit zu unterbieten (was ich nicht glaube), so schlage ich ihm als Verbesserung den Befehl MULS #999,D0 mit nur noch 50 Takten vor. Hier dürfte dann auch der letzte Experte seinen Löffel abgeben. Ein weiteres interessantes Untersuchungsobjekt sind die Bitmanipulationsbefehle. Die Befehle

```
BSET #1,D0
```

und

```
BSET #21,D0
```

benötigen 10 bzw. 12 Takte. Bei genauerer Untersuchung ergibt sich dann das folgende Bild:

```
BSET #xx,D0
```

benötigt 10 Takte falls xx im Bereich von 0-15 liegt und 12 Takte für xx von 16-31. Eine wirkliche Überraschung erlebt man bei der Untersuchung von Befehlen wie

```
MOVE.B $BFDD00,D0
MOVE.L $BFDD00,D0
```

und

```
MOVE.L $BFDD00,$BFDD00.
```

Anstatt der erwarteten Werte 16,20 und 36 Takte erhält man nach Überprüfung mit dem Testprogramm 30, 40 und 70 Takte. Der Amiga scheint also diese Befehle auf Vielfache von 10 aufzurunden. Fügt man noch ein, zwei oder drei NOP-Befehle hinzu, so steigt die Verwunderung noch.

Die Erklärung: der 68000-Prozessor schaltet beim Zugriff auf den Peripheriebaustein 8520 in den synchronen Modus um, und da dieser Baustein mit einem Zehntel der normalen Taktfrequenz betrieben wird, kommt es zu diesen merkwürdigen Zehnersprüngen bei den Ausführungszeiten.

Zum Programm: Als erstes werden alle äußeren Einflüsse unterbunden, indem die Interrupts und DMA-Zugriffe abgeschaltet werden. Würde man diese nicht abschalten, so könnte es zu Unregelmäßigkeiten im Ablauf kommen. So kann z.B. ein Interrupt genau während des Ablaufs der Testschleife auftreten, und wir würden dessen Ausführungszeit mit-

```
;******** Taktzyklen bestimmen *********
;*-------------------------------------*
;* Prog.name:     taktik.s             *
;*-------------------------------------*
;* Autor:         Gerald Steffens      *
;* Copyright:     bei mir              *
;* Assembler:     A68k                 *
;* Hardware:      512K,Kickstart1.2    *
;***************************************

ExecBase:         equ     4
OldOpenLibrary:   equ  -408
CloseLibrary:     equ  -414
Output:           equ   -60
Write:            equ   -48
Disable:          equ  -120
Enable:           equ  -126
RawDoFmt:         equ  -522
Umlauf:           equ    99  ; für 100 Umläufe

start:           equ      0  ; Register d6 läuft
ende:            equ     20  ; von start bis ende

; hängt die Testschleife nicht von d6 ab, reicht start=ende=0


;***** hier beginnt der Spaß *****
    move.l   ExecBase,a6         ; es wird
    lea      dosname,a1          ; die Dos-Library
    jsr      OldOpenLibrary(a6); geöffnet
    move.l   d0,dosbase          ; ging alles gut
    beq      abgang              ; nein, dann Abgang

    move.l   d0,a6               ; DosBase
    jsr      Output(a6)          ; Output-handle
    move.l   d0,handle           ; Ausgabe-Datei
```

»Taktik.s«: Ein Programm, um Taktzyklen von Routinen zu messen und Programme zu optimieren (Anfang)

stoppen. Oder die Bildschirm-DMA stiehlt unserem armen Prozessor einige Zyklen und wir erhalten falsche Ergebnisse. Danach wird der Timer A des CIA-B initialisiert und gestartet. Hiermit wird die Zeit für die Testschleife gestoppt. Die Differenz zwischen der Zeit für die Testschleife und einer Leerschleife ergibt dann genau die für die Ausführung der Testbefehle benötigte Zeit.

Messen Sie kritische Routinen

Da die Testschleife 100mal durchlaufen wird, der Zähler aber nur mit einem Zehntel des Systemtakts betrieben wird, erhalten wir nach einer Division durch 10 die gesuchte Taktzahl. Anschließend wird nur noch das Ergebnis ausgegeben.

Das Programm wurde komplett mit dem PD-Assembler A68k geschrieben, ist sehr kurz und läßt sich schnell starten. Es sollte auf jedem anderen Assembler ebenfalls ohne Änderungen laufen.

Man gibt in der Testschleife seinen Befehl oder aber seine zu testende Befehlsfolge ein, assembliert und linkt mit »A68k taktik.s« und »Blink taktik.o« das Programm, und hat so schon nach dem Aufruf von »Taktik« die gesuchte Taktzahl auf dem Bildschirm.

Solange die Testbefehle weniger als 6000 Takte verbrauchen (und das ist eine ganze Menge), treten keine Probleme auf. Da der Timer aber mit Wortbreite arbeitet, kann es bei noch längeren Befehlsfolgen zu einem Überlauf kommen. Durch das Abschalten des gesammten DMA (direct memory access) kann es bei längeren Befehlsfolgen zu einem Flackern des Monitorbildes kommen.

Achtung: Das Programm paßt nicht unbedingt in die Multitasking-Umgebung, da es alle Interrupts und DMA-Vorgänge sperrt. Es handelt sich um einen echten Hardwarehack und man sollte am besten keine wichtigen Arbeiten parallel zu diesem Programm laufen lassen, insbesondere der Disketten- und Festplattenbetrieb sollten unterbleiben, wenn man keinen Datenverlust erleiden möchte. Dieses Tool ist eher für Profis gedacht, Anfänger sollten die Finger davon lassen. Die Benutzung des Programms erfolgt auf eigene Gefahr, der Autor übernimmt keinerlei Haftung bei irgendwelchen Schäden.

Keine Angst, ich hatte noch keinerlei Probleme mit dem Programm!! Ansonsten viel Spaß. Noch ein Tip: Assembler und Programm hält man am besten im RAM und ist so mit wenigen Tastendrücken vom Editor beim lauffähigen Programm (und umgekehrt). ■

```
   move.l   #start,count
loop:
   move.l   ExecBase,a6         ; Interrupts
   jsr      Disable(a6)         ; sperren
warte:
   move.l   $dff004,d7          ; um ein Flackern
   andi.l   #$0001ffff,d7       ; des Monitor zu
   lsr.l    #8,d7               ; unterbinden wird
   cmpi.l   #310,d7             ; auf Zeile 310
   bne.s    warte               ; gewartet

   move     #$0200,$dff096      ; DMA aus
   move.l   count(pc),d6        ;
   move     #Umlauf,d7
   lea      $bfd000,a6          ; Hardwarebasis
   move.b   #$00,$e00(a6)       ; Timer A stop
   move.b   #$ff,$400(a6)       ; Low
   move.b   #$ff,$500(a6)       ; High  und speichern
   move.b   #$01,$e00(a6)       ; start

;************* Testschleife *************
;*                                      *
;*          d0-d5 und a0-a6             *
;*    stehen zur freien Verfügung       *
;* d6 durchläuft die Werte start - ende *
;*    d7 nicht benutzen, weil Zähler    *
;****************************************

testloop:

;          *** hier Testbefehle einfügen ***

;          *** hier Testbefehle einfügen ***

;          *** hier Testbefehle einfügen ***

     mulu    d6,d0

; weitere interessante Befehle und passende Werte

;     mulu    #640,d0         ;  0-0
;     bset    d6,d0           ;  0-20
;     lsl     d6,d0           ;  0-70
;     move.b   $bfdd00,d0

;*********************************

   dbra     d7,testloop

   lea      $bfd000,a6
   move.b   #$00,$e00(a6)       ; Timer A stop
   move     $500(a6),d7         ; High holen
   move.b   $400(a6),d7         ; Low holen
   move     #$8200,$dff096      ; DMA ein
   movem.l  d0-d5/a0-a6,-(a7)   ; sichern
   move.l   ExecBase,a6         ; Interrupts
   jsr      Enable(a6)          ; erlauben
```

```
   addi     #107,d7             ; Zeit einer Leerschleife
   neg      d7                  ;
   add      #2,d7               ; Korrektur ( Rundung )
   ext.l    d7
   divu     #10,d7              ; Taktzahl justieren
   lea      datas,a1            ; Puffer für die Werte
   move.l   count(pc),(a1)+     ; Zählerwert
   ext.l    d7                  ; und
   move.l   d7,(a1)             ; Taktzahl
                                ; eintragen
   move.l   ExecBase,a6
   lea      formatstr(pc),a0    ; Formatstring
   lea      datas(pc),a1        ; Data-Puffer
   lea      prog(pc),a2         ; Hilfsroutine
   lea      buffer(pc),a3       ; Ausgabepuffer
   jsr      RawDoFmt(a6)        ; Werte verarbeiten
   move.l   handle(pc),d1       ; Output-handle
   move.l   #buffer,d2          ; Ausgabepuffer
   move.l   textend(pc),d3
   sub.l    #buffer,d3          ; textlänge
   move.l   dosbase(pc),a6
   jsr      Write(a6)           ; Text ausgeben
   movem.l  (a7)+,d0-d5/a0-a6   ; restaurieren

   addq.l   #1,count            ; Zähler + 1
   move.l   count(pc),d6        ;
   cmpi.l   #ende,d6            ; Zähler<=ende?
   ble      loop                ; weiter->
   move.l   dosbase(pc),a1      ; Dos-Library
   move.l   ExecBase,a6         ; wieder
   jsr      CloseLibrary(a6)    ; schließen

abgang:
   moveq    #0,d0
   rts

; *** Hilfsroutine *** wird von RawDoFmt aufgerufen
prog:
   move.b   d0,(a3)+
   clr.b    (a3)
   move.l   a3,textend
   rts

count:      dc.l  0
dosbase:    dc.l  0
handle:     dc.l  0
textend:    dc.l  0
datas:      dc.l  0
            dc.l  0
buffer:     ds.b  100
dosname:    dc.b  "dos.library",0
   even
formatstr:  dc.b  "nr: %8ld  takte= %5ld",10,0

   end        © 1993 M&T
```

»Taktik.s«: Ein Programm, um Taktzyklen von Routinen zu messen und Programme zu optimieren

Systemfunktionen umlenken

Patch und Purge

Verbotene Sachen sind immer reizvoll. Und eigentlich gehört das Umbiegen von Systemfunktionen zu den übelsten Techniken in einem Multitaskingsystem wie dem Amiga. Dennoch hier ein schönes und praktisches Beispiel, wie man die DOS-Funktionen manipuliert, um z.B. mehr Sicherheit in seine Datensammlungen zu bekommen.

von Dietmar Craul

Wir stellen Ihnen im folgenden zwei Programme vor, die Ihnen die Versionsverwaltung von Programmen erheblich vereinfachen. Gerade für Programmierer sind beide Tools nützlich, da es immer wichtig ist, die aktuellste Version eines Programmtexts etc. zu kennen.

Das erste Programm »VPatch.c« ändert (»patcht«) die »DOS.library« derart, daß bei einem Überschreibversuch einer Datei immer neue Versionen erzeugt werden und ein Leseversuch auf die Datei grundsätzlich auf die höchste Version erfolgt (Ausnahmen siehe Detailbeschreibung). Das zweite Programm »Purge.c« erleichtert den Umgang mit den unterschiedlichen Dateiversionen nochmals: Es bearbeitet den Inhalt von einzelnen Verzeichnissen bis zu kompletten Volumes und löscht alle alten Versionen einer Datei, wobei immer die älteste gefundene Version in eine »001«-Version umbenannt wird.

Anwendung des »VPatch«-Programms:

»VPatch« ist von CLI/Shell lauffähig und erwartet als Parameter den Namen einer »Extensionsdatei«. In ihr stehen die Dateiendungen, für die Versionen erzeugt werden sollen. In der Datei muß für jede Endung eine neue Zeile verwendet werden. Beispiel :

```
.c
.txt
.asc
```

Zur Anpassung des Programms an eigene Wünsche muß man eine Textdatei (z.B. mit Ed, MEmacs, etc.) erstellen und dort die Dateiendungen eintragen, für die Versionen erstellt werden sollen. Jede Endung darf max. 79 Zeichen umfassen.

Das Programm sollte mit run/runback aus der Shell gestartet werden, um die Shell wieder freizugeben. Der Patch wird aufgehoben, indem man VPatch nochmals startet.

Der Patch berücksichtigt nur Dateien mit einer bestimmten Endung (s.o). Wird so eine Datei gesichert, fügt der Amiga eine neue Versionsnummer in den Dateinamen ein und zwar nach folgendem Muster:

`<Dateiname>.<Version>.<Endung>`

Die Version wird eingefügt, da manche Programme die Endung des Dateinamens benötigen (z.B C-Compiler => ».c«).

Man kann auch gezielt bestimmte Versionen überschreiben, wenn man die Versionnummer, wie dargestellt, in den Dateinamen einfügt.

Will man neue Versionen erzeugen, muß man immer den gleichen Dateinamen ohne Versionsnummer angeben (z.B. »test.c«).

Um eine Datei mit mehreren Versionen zu lesen, gibt es zwei Möglichkeiten:

– man gibt die Versionsnummer, wie beschrieben, an – diese Version wird geladen.
– man gibt den ursprünglichen Dateinamen (ohne Versionsnummer) an, die höchste vorhandene Version wird geladen.

Falls manche Programme beim Speichern abfragen, ob Sie die bestehende Datei ersetzen wollen, ist das zu bestätigen, es wird trotzdem eine neue Version erzeugt.

Das Patchen erweist sich besonders bei evolutionären Entwicklungen (Quellcodes, Layouts, 3-D-Objekte) als nützlich.

Anmerkung: Zu jeweils allen Versionen einer Datei wird eine Hilfsdatei »<name>.vers« angelegt, welche die höchste vorhandene Versionsnummer enthält. Falls alte Dateiversionen gelöscht werden sollen, sollte man nie den »delete«-Befehl, sondern immer das »Purge«-

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include <exec/exec.h>
#include <intuition/intuition.h>
#include <proto/dos.h>
#include <proto/exec.h>
extern BPTR __asm newopen(register __d1 char *,register __d2 long);
extern BPTR __asm newlock(register __d1 char *,register __d2 long);
void main(int,char **);
extern struct DosLibrary *DOSBase;
extern struct ExecBase   *SysBase;
APTR oldOpen,oldLock;
/* Struktur, um Verwaltungsdaten für Extensionen aufzunehmen */
typedef struct
{ char extension[80];
  long length;
} VINFO;
VINFO *info;
long anzahl;
void main(int argc,char **argv)
{ struct IntuiMessage send,*receive;
  struct MsgPort      *port;
  BPTR file;
  char buffer[80];
  int i=0;
  VINFO *inf;
  /* Message-Port "VPort" vorhanden? Dann Msg Klasse 0xffffffff senden */
  if (port=FindPort("VPort"))
  { send.Class=0xffffffff;
    PutMsg(port,(struct Message*)&send);
    FPuts(Output(),"DOS-Open() u. DOS-Lock() Patch aufgehoben\n");
    return;
  }
  else
  { if (argc!=2)
    { FPuts(Output(),"FORMAT : vpatch <Parameterdatei>\n");
      return;
    }
   /* angegebene Parameterdatei einlesen */
   if (file=Open(argv[1],MODE_OLDFILE))
   { while (FGets(file,buffer,79))
     { if (strcmp(buffer,"\n"))         /* Keine Leerzeile */
       { inf=info;                      /* "realloc()" von Hand ... */
         if (info=AllocVec((i+1)*sizeof(VINFO),MEMF_CLEAR|MEMF_PUBLIC))
         { CopyMem(inf,info,i*sizeof(VINFO));
          FreeVec(inf);          /* ... und dann Daten einkopieren */
          strcpy((*(info+i)).extension,buffer);
          (*(info+i)).length=strlen(buffer)-1;
          (*(info+i)).extension[(*(info+i)).length]='\0';
          i++;
         }
         else
         { FreeVec(info);
           FPuts(Output(),"Kein freier Speicher für Extensionsliste\n");
           Close(file);
           return;
     } } }
     anzahl=i;
 /* Message-Port einrichten, Vektoren verbiegen und auf spezielle
  Message warten, dann Vektoren wiederherstellen und Programm beenden. */
    if (port=CreateMsgPort())
    { port->mp_Node.ln_Name="VPort";
      AddPort(port);
```

»Patch.c«: Mit diesem Tool ändern Sie die DOS.library, um automatisch Sicherheitskopien abzulegen (Anfang)

Programm verwenden. Durch die Möglichkeit, selektiv Dateiversionen zu erstellen, bleibt dem Benutzer die größtmögliche Entscheidungsfreiheit im Versionsmanagement, außerdem gibt es keine Probleme, wenn Devices (z.B. »CON:«, oder »PRT:«) mit »Open()« geöffnet werden. VPatch sollte nie während einer I/O-Operation gestartet werden!

Funktion des VPatch-Programms:

Das Programm richtet, falls noch nicht vorhanden, einen Message-Port ein, der dazu dient, das Programm nach nochmaligem Aufruf zu beenden, andernfalls wird durch diesen Message-Port an das bereits laufende Patch-Programm eine Message verschickt, welche die Beendigung des laufenden Programms veranlaßt. Danach wird die Parameterdatei ausgelesen und die darin vermerkten Endungen und ihre Länge in einem Strukturarray festgehalten.

Als zweites werden mit »SetFunction()« die Betriebssystemfunktionen auf die eigenen Routinen »verbogen«, dann wartet das Programm auf das Eintreffen der Message. Daß das Programm im Speicher steht, vereinfacht die Aufhebung des Patches und ist unter C auch einfacher, als die Routinen in einen reservierten Speicherbereich zu kopieren (-> Linker !).

Die Funktion »newOpen()« stellt zuerst fest, ob der Dateiname eine der gültigen Endungen hat. Ist das der Fall und eine neue Datei soll erzeugt werden (MODE_NEWFILE), wobei keine Versionnummer angegeben ist, ermittelt das Programm die neue, höchste Versionsnummer, verändert den Dateinamen und speichert die höchste Versionsnummer in einer Hilfsdatei.

Der Lesevorgang (MODE_OLDFILE) erfolgt ähnlich: Die Funktion liest die höchste Versionsnummer aus der Hilfsdatei und verändert den Dateinamen entsprechend, vorausgesetzt, es wurde keine Version angegeben. In allen anderen Fällen wird der betreffende Dateiname nur »durchgestellt«.

Aus dieser Vorgehensweise ergibt sich auch ein Problem: Der Patch kann keine Versionen von Dateien verwalten, die von Programmen erzeugt werden, die eine Datei im sog. Modus MODE_READWRITE öffnen, weil hier die Aktion nicht einwandfrei bestimmbar ist.

Die »newLock()«-Funktion arbeitet ähnlich, sie versucht grundsätzlich (unabhängig von der Dateiendung), zu einer Datei die höchste vorhandene Version zu ermitteln und ein Lock zu erhalten (wieder vorausgesetzt, es wurde keine Versionsnummer angegeben), sonst gleicht die Arbeitsweise der »newOpen()«-Funktion.

Auch aus diesem Ablauf ergibt sich ein Problem: Versucht ein Programm, auf eine gerade geschriebene Datei, bei der Versionen erzeugt werden sollen, ein Lock zu bekommen,

```
      oldOpen=SetFunction((struct Library*)DOSBase,-30,newopen);
      oldLock=SetFunction((struct Library*)DOSBase,-84,newlock);
      FPuts(Output(),"DOS-Open() u. DOS-Lock() gepatcht\n");
      Close(file);
      do
      { receive=(struct IntuiMessage*)WaitPort(port);
      } while (receive->Class!=0xffffffff);
      FreeVec(info);
      RemPort(port);
      DeleteMsgPort(port);
      SetFunction((struct Library*)DOSBase,-30,oldOpen);
      SetFunction((struct Library*)DOSBase,-84,oldLock);
  } }
  else if (!file)
    FPuts(Output(),"Parameterdatei nicht vorhanden\n");
 }
 return;
}
extern BPTR __asm newopen(register __d1 char *name,register __d2 long mode)
{ char newname[256],hilfe[256],*p;
  BPTR fh;
  short x=0,len,kenn=-1,pos=0,in=0;
  /* Zugriff auf globale Daten, Libraryadressen etc. */
  geta4();
  strcpy(newname,name);
  strcpy(hilfe,name);
  strcat(hilfe,".vers");
  len=strlen(name);
  /* Jetzt Open() wieder auf DOS-Library umstellen */
  SetFunction((struct Library*)DOSBase,-30,oldOpen);
  /* um welche Extension handelt es sich ? */
  for (x=0;x<anzahl && kenn==-1;x++)
  { if (!strnicmp(newname+(len-(*(info+x)).length),(*(info+x)).extension,
        (*(info+x)).length))
    { pos=len-(*(info+x)).length;
      kenn=x;
  } }
  /* Neue Datei => Wenn keine Version angegeben, höchste lesen */
  if (mode==MODE_NEWFILE && kenn!=-1)
  { if (p=stpchr(newname,'.'))
    {  /* Versionsnummer vorhanden ? */
      if (!(x=atoi(p+1)))
      { if (fh=Open(hilfe,MODE_OLDFILE))
        { FRead(fh,&x,sizeof(x),1);
          Seek(fh,0,OFFSET_BEGINNING);
        }
        else
        { if (!(fh=Open(hilfe,MODE_NEWFILE)))
          x=1000;
        }
        /* wenn Vers.-Nummer <1000 => Zusatzdatei 'updaten' */
        if (x<999)
        { in=1;
          x++;
          FWrite(fh,&x,sizeof(x),1);
          Close(fh);
        }
        else
          return((BPTR)0);
  } } }
  /* wie bei MODE_NEWFILE, nur kein update auf Zusatzdatei */
  else if (mode==MODE_OLDFILE && kenn!=-1)
  { if (p=stpchr(newname,'.'))
    { if (!(x=atoi(p+1)))
      { if (fh=Open(hilfe,MODE_OLDFILE))
        { in=1;
          FRead(fh,&x,sizeof(x),1);
          Close(fh);
  }}  } }
  /* Neue Dateinamen mit Versionsnamen erstellen */
  if (x && kenn!=-1 && in)
  { sprintf(newname+pos,".%03d",x);
    strcpy(newname+pos+4,(*(info+kenn)).extension);
  }
  fh=Open(newname,mode);
  /* Datei öffnen, Vektor auf 'newopen' verbiegen, BPTR zurückgeben */
  SetFunction((struct Library*)DOSBase,-30,newopen);
  return(fh);
}
extern BPTR __asm newlock(register __d1 char *name,register __d2 long mode)
{ BPTR lock,fh;
  char newname[256],hilfe[256],vname[256],*p;
  short version;
  /* Zugriff auf globale Daten, Libraryadressen etc. */
  geta4();
  strcpy(newname,name);
  strcpy(vname,name);
  /* Wenn keine Version angegeben, höchste lesen */
  if (p=stpchr(newname,'.'))
  {  /* Versionsnummer vorhanden ? */
   if (!(version=atoi(p+1)))
   { strcpy(hilfe,p);
     strcat(vname,".vers");
     /* originale Open()-Funktion, dann Versuch Zusatzdatei mit
      Versionsnummer zu lesen       */
     SetFunction((struct Library*)DOSBase,-30,oldOpen);
     if (fh=Open(vname,MODE_OLDFILE))
     { FRead(fh,&version,sizeof(version),1);
       Close(fh);
       /* Versionsnummer in Dateiname einfügen */
       sprintf(p,".%03d",version);
       strcat(newname,hilfe);
     }
     SetFunction((struct Library*)DOSBase,-30,newopen);
  } }
  /* Lock()-Funktion ausführen den Lock ermitteln, den Vektor wieder
    verbiegen und den Lock zurückgeben   */
  SetFunction((struct Library*)DOSBase,-84,oldLock);
  lock=Lock(newname,mode);
  SetFunction((struct Library*)DOSBase,-84,newlock);
  return(lock);
}
```


»Patch.c«: Mit diesem Tool ändern Sie die DOS.library, um automatisch Sicherheitskopien abzulegen (Ende)

erhält das Programm ein Lock auf die höchste Version. Mittels »Examine()« auf diesen Lock wird dann der Dateiname (mit Versionsnummer) ermittelt. Übernimmt das Programm diesen Dateinamen für weitere Schreiboperationen, können ebenfalls keine Versionen erstellt werden, da die Versionsnummer in dem Dateinamen explizit angegeben wurde. »Beckertext-II« arbeitet z.B. auf diese Art.

Der Patch wurde mit folgenden Programmen erfolgreich getestet: LSE (Lattice Screen Editor), ED, MEmacs Imagine 2.0, Pagestream 2.2 und GFA-Basic 3.5.

Anwendung des »Purge«-Programms:

Das Purge-Programm dient dazu, überflüssige Dateiversionen zu entfernen. Es kann ausschließlich von CLI/Shell aus gestartet werden.

Die Aufrufsyntax :

- `purge [-r] <Datei/Pfadname>`

Der Parameter »-r« veranlaßt das Programm, alle Verzeichnisse unter dem angegebenen Pfad zu durchsuchen. Der Parameter »<Datei/Pfadname>« bestimmt die Versionsgruppen, deren alte Versionen entfernt werden sollen. Als Name können Wildcards entspr. den Amiga-DOS-Konventionen (OS 2.0) verwendet werden.

Die bisher höchste Versionsnummer wird dann zur Version ».001« umbenannt, z.B.:

- `purge df0:test.c` -> löscht alle Versionen von test.c im Stammverzeichnis von df0:
- `purge df0:#?.c` -> löscht alle alten Versionen von Dateien im Stammverzeichnis von df0:, mit Endung ».c«.
- `purge -r dh0:#?` -> löscht in allen Verzeichnissen von DH0: alle alten Versionen.

Funktion des »Purge«-Programms:

Das angegebene Verzeichnis wird Eintrag für Eintrag gelesen, bei einer Übereinstimmung mit dem Dateinamen/-muster wird versucht, die Hilfsdatei zu finden, alle alten Versionen zu löschen und die älteste Version umzubenennen, ebenso erfolgt ein Update der Hilfsdatei.

Ist der Vorgang erfolgreich, wird das Verzeichnis erneut gelesen, und wie oben verfahren, bis keine Einträge mehr vorhanden sind.

Ist das Rekursiv-Flag gesetzt und stößt die Funktion »scandir()« auf ein Unterverzeichnis, erfolgt ein rekursiver Aufruf mit dem um das Unterverzeichnis ergänzten Pfad. Dort wird das Verzeichnis erneut abgearbeitet. ■

Auf der Diskette zum Heft finden Sie zwei komplette Projektverzeichnisse für den Lattice-C-Compiler. Dies ermöglicht Ihnen, die Programme jederzeit neu zu kompilieren, bzw. die Compileroptionen zu erkennen. Beide Programme sind lauffähig ab OS 2.0.

```
#include <stdio.h>
#include <string.h>
#include <exec/exec.h>
#include <proto/dos.h>
#include <proto/exec.h>
void main(int,char **);
short __regargs purge(char *,char *);
void __regargs scandir(char *,char *);
short recur;
void main(int argc,char **argv)
{ static char path[256],node[256],*filep;
  long len;
  if (!argc)
   return;
  if (argc==2 && argv[1][0]=='?')
  { FPuts(Output(),"USAGE : purge [-r] <file>\n");
    return;                                          }
  if (argc==3 && argv[1][0]=='-' && argv[1][1]=='r')
   recur=1;
  if (filep=FilePart(argv[argc-1]))  /* Pfad und Pattern trennen */
  { len=filep-argv[argc-1];
    strncpy(path,argv[argc-1],len);
    strcpy(node,filep);
    scandir(path,node);                              } }
void __regargs scandir(char *path,char *pattern)
{ static char buffer[516],left[256],right[256];
  char *buffer1,*p;
  struct FileInfoBlock *fib;
  BPTR lock;
  short success=1,patt,purged=0;
  /* Zwischenspeicher allokieren */
  if (!(buffer1=(char *)AllocVec(256,MEMF_CLEAR|MEMF_PUBLIC)))
  { FPuts(Output(),"Nicht genug Speicher\n");
    return;                                            }
  if (!(fib=(struct FileInfoBlock*)AllocVec(sizeof(struct
           FileInfoBlock),MEMF_CLEAR|MEMF_PUBLIC)))
  { FPuts(Output(),"Nicht genug Speicher\n");
    FreeVec(buffer1);
    return;                                            }
  patt=ParsePattern(pattern,buffer,512);
  if (lock=Lock(path,ACCESS_READ))
  { if (Examine(lock,fib))
    { while (success)  /* Verzeichnis auf Matches prüfen */
      { success=ExNext(lock,fib);
        /* Pattern vorhanden ==> Dateinamen vergleichen */
        if (patt && fib->fib_DirEntryType<=0 && MatchPattern(buffer,
            fib->fib_FileName))
        { if (p=stpchr(fib->fib_FileName,'.'))
        { /* Wenn Punkt im Dateinamen, dann Namenstamm abtrennen
             und versuchen, diese Gruppe zu löschen */
         memset(left,0,sizeof(left));
         strncpy(left,fib->fib_FileName,p-(fib->fib_FileName));
         strcpy(right,p+4);
         strcat(left,right);
         if (purge(path,left))
         { UnLock(lock); /* Dateien gelöscht, Dir muß "gelockt" werden */
           if (lock=Lock(path,ACCESS_READ))
           { Examine(lock,fib);
             success=ExNext(lock,fib); /* ExNext für Volume */
    } } } }
 /* Name ohne Pattern angegeben und noch kein purge auf das Verzeichnis */
        else if (fib->fib_DirEntryType<=0 && !patt && !purged)
        { if (purge(path,pattern))
          { purged=1;
            UnLock(lock); /* Dateien gelöscht ==> neuer Lock */
            if (lock=Lock(path,ACCESS_READ))
           { Examine(lock,fib);  /* ExNext für Volume */
             success=ExNext(lock,fib);                       }} }
      /* Unterverzeichnis und "-r"-Flag angegeben */
       if (fib->fib_DirEntryType>0 && recur)
       { strcpy(buffer1,path);          /* Pfad retten */
         strcat(path,fib->fib_FileName);  /* neuer Pfad */
         strcat(path,"/");            /* Slash dazu */
         UnLock(lock);              /* altes Lock freigeben */
         scandir(path,pattern);        /* und Rekursion */
         strcpy(path,buffer1);          /* alten Pfad zurück */
         if (!(lock=Lock(path,ACCESS_READ))) /* und "relock" */
         return;                                           } } }
    UnLock(lock);                                          }
  FreeVec(buffer1); /* Speicher freigeben */
  FreeVec(fib);                                             }
short __regargs purge(char *path,char *name)
{ BPTR fh;
  short x,max,purged=0;
  static char substring[256],string[256],help[256],*p;
  strcpy(string,path);
  strcat(string,name);
  strcpy(help,string);
  strcat(help,".vers");
  if (p=stpchr(string,'.'))
  { strcpy(substring,p);
    if (fh=Open(help,MODE_OLDFILE)) /* Versionsdatei öffnen */
    { FRead(fh,&max,sizeof(max),1); /* höchste Versionsnummer */
      if (max>1)
      { for (x=max-1;x>0;x--)
        { purged=1;
          sprintf(p,".%03d",x); /* Dateinamen mit Versionsnr. erstellen */
          strcat(string,substring);
          string[strlen(string)]=0;
          DeleteFile(string);   /* Version "x" der Datei löschen */
          strcpy(help,string);
          strcat(help," - purged\n");
          FPuts(Output(),help);                         }
        sprintf(p,".%03d",max);
        strcat(string,substring);
        strcpy(help,string);
        *p='.';
        *(p+1)='0';
        *(p+2)='0';
        *(p+3)='1';
        Rename(help,string); /* und höchste Version umbenennen */
        max=1;         /* Versionsdatei neu schreiben ("1") */
        Seek(fh,0,OFFSET_BEGINNING);
        FWrite(fh,&max,sizeof(max),1);                   }
      Close(fh);                                     } }
  return(purged);
}
```


»Purge.c«: Mit diesem hilfreichen Tool bringen Sie zusätzlich Ordnung in den Dateiendschungel

Zufallsgenerator für die Shell

Programmierte Fügung

Egal, ob es sich um eine unerwartete Gehaltserhöhung, sechs Richtige im Lotto oder um die Bekanntschaft einer jungen Dame beim Einkaufen handelt, Zufälle bestimmen und bereichern das Leben ungemein.

von Stephan Gromer

Am Computer – genauer auf DOS-Ebene – mußten wir bisher auf einen Luxus verzichten: Eine einmal geschriebene Batch-Datei verrichtet ihre Arbeit immer auf dieselbe Art. Was man ja zugegebenermaßen auch von ihr erwartet ....

Aber es gibt Fälle, in denen man sich ein gewisses Maß an kontrolliertem Chaos wünscht: Beispielsweise wenn Sie verschiedene Bildschirmschoner oder Workbenchhintergrundgrafiken haben, bei denen Sie die Entscheidung, welchen bzw. welche Ihr Amiga verwenden soll, lieber Ihrer »Startup-Sequence« überlassen möchten – bisher kaum möglich.

Hier hilft das Assembler-Programm (Seka) »Random.s«: Es liefert eine Zufallszahl in einem von Ihnen vorgegebenen Intervall. Diese Zufallszahl kann in einer Environmentvariable gespeichert und mittels IF-ELSE-ENDIF-Konstrukten entsprechend verwendet werden.

»Random« benötigt als Parameter lediglich eine Zahl als obere Intervallgrenze. Diese kann im Bereich von 1 bis 255 liegen. Random liefert dann eine Zufallszahl im Bereich von 0 bis zu der angegebenen Grenze minus eins. So kann Random 5 folgende Werte liefern:

```
000 001 002 003 004
```

Ein einfaches Anwendungsbeispiel zeigt die obige Batchdatei »Bsp.bat«.

```
Random >ENV:zufall 5
If $zufall EQ "000"
 Echo "Hier Prog. f. Zahl 0 aufrufen"
EndIf
If $zufall EQ "001"
 Echo "Hier Prog. f. Zahl 1 aufrufen"
EndIf
If $zufall EQ "002"
 Echo "Hier Prog. f. Zahl 2 aufrufen"
EndIf
If $zufall EQ "003"
 Echo "Hier Prog. f. Zahl 3 aufrufen"
EndIf
If $zufall EQ "004"
 Echo "Hier Prog. f. Zahl 4 aufrufen"
EndIf
```


»Bsp.bat«: So setzen Sie den Zufallsgenerator in Batch-Dateien ein

Wollen Sie die Zufallszahl nicht nur auf dem Bildschirm sehen, sondern in einer Variable speichern, müssen Sie das wie folgt angeben:

```
Random >ENV:Zufallsvariable 5
```

Das ist eine einfache Umleitung der Ausgabe von Random in die Datei Zufallsvariable im Verzeichnis »ENV:«. Das CLI verlangt, daß diese Umleitung als erster Parameter nach dem Programmnamen folgt. Damit das CLI die Datei als Variable erkennen kann, muß das Zielverzeichnis »ENV:« sein.

Random liefert das Ergebnis immer als dreistellige Ziffer, ggf. mit führenden Nullen. Mit »Random ?« oder bei fehlerhafter Angabe der oberen Grenze erhalten Sie eine Kurzbeschreibung des Befehls. Im letzteren Falle liefert Random den Wert FAIL ans CLI zurück.

Die Variable kann dann, wie aus dem Beispiel ersichtlich, mit einer Konstruktion wie

```
If $Zufallsvariable EQ "000"
 ; Was soll getan werden, wenn 0?
EndIf
```

verarbeitet werden.

Das »$«-Zeichen vor dem Variablennamen signaliesert dem IF-Befehl, daß er hier die Variable Zufallsvariable aus dem »ENV:«-Verzeichnis überprüfen soll.

Zum Programm: Nach dem Öffnen der »Dos.library« werden die Parameter geprüft und ggf. eine Fehlermeldung gedruckt (Diese würde übrigens auch in der Variable landen). Ging alles glatt, wird die Position des Elektronenstrahls gelesen und aus dieser eine Zufallszahl im erlaubten Bereich berechnet und diese ausgegeben. (Es ist hier nicht möglich eine »mathematische« Zufallszahlenroutine zu verwenden, da Random ja nur eine Zufallszahl liefert und dann beendet wird. Das Programm würde dann immer nur die gleiche Zufallszahl liefern.) Achtung: Ältere Versionen von IF können zum Teil nichts mit Environmentvariablen anfangen. In diesem Fall müßten Sie sich eine neuere Version besorgen. ■

```
;* Random [>Dest:] Max/S von Stephan Gromer
ExecBase      =  4
OldOpenLibrary=-408
CloseLibrary  =-414
Output        =- 60
Write         =- 48
RETURN_OK     =  0
RETURN_FAIL   =  20
main:
 movem.l a0/d0,-(sp)
 move.l ExecBase.w,a6
 lea    dosname(pc),a1
 jsr    OldOpenLibrary(a6)
 move.l d0,a6
 movem.l (sp)+,a0/d1
 subq.w #1,d1
 bne.S  weiter
error:
 moveq #RETURN_FAIL,d4
infoout:
 move.l #infotext,d2              ; D2:^Text
 moveq  #infotextende-infotext,d3; D3: Länge
textout:
 jsr    Output(a6)
 move.l d0,d1             ; D1:^Filehandle
 beq.S  cant_write
 jsr    Write(a6)
cant_write:
 move.l a6,a1
 move.l ExecBase.w,a6
 jsr    CloseLibrary(a6)
 move.l d4,d0
 rts
weiter:
 moveq  #RETURN_OK,d4
 moveq  #0,d0; Ergebnis init.
loop:
 tst.w  d1
 beq.S  check
 cmp.b  #'?',(a0)
 beq.S  infoout
 subq.w #1,d1
 cmp.b  #' ',(a0)+
 beq.S  loop
 sub.b  #'0',-1(a0)
 bmi.S  error
 cmp.b  #10,-1(a0)
 bpl.S  error
 mulu   #10,d0
 add.b  -1(a0),d0
 bra.S  loop
check:
 tst.l  d0
 beq.S  error
 cmp.w  #256,d0
 bpl.S  error
 moveq  #0,d1
 move.w $dff006,d1; Vertical Beam Position
 divu   d0,d1
 clr.w  d1
 swap   d1
 lea    randomtext(pc),a0
 move.l a0,d2
D2:^Text
 moveq  #randomtextende-randomtext,d3; Länge
 divu   #100,d1
 add.b  d1,(a0)+
 clr.w  d1
 swap   d1
 divu   #10,d1
 add.b  d1,(a0)+
 swap   d1
 add.b  d1,(a0)
 bra    textout
dosname: dc.b "dos.library",0
randomtext:
 dc.b "000",$a,$d
randomtextende:
infotext:
 dc.b "Random Max/S",$a,$d
 dc.b "Returns a randomvaluestring ranging"
 dc.b " from 0 to Max-1",$a,$d
 dc.b "Max can range from 1 to 255",$a,$d
infotextende:             ;
```


»Random.s« erzeugt Zufallszahlen mit Hilfe der Rasterstrahlposition

Intuition-Programmierung

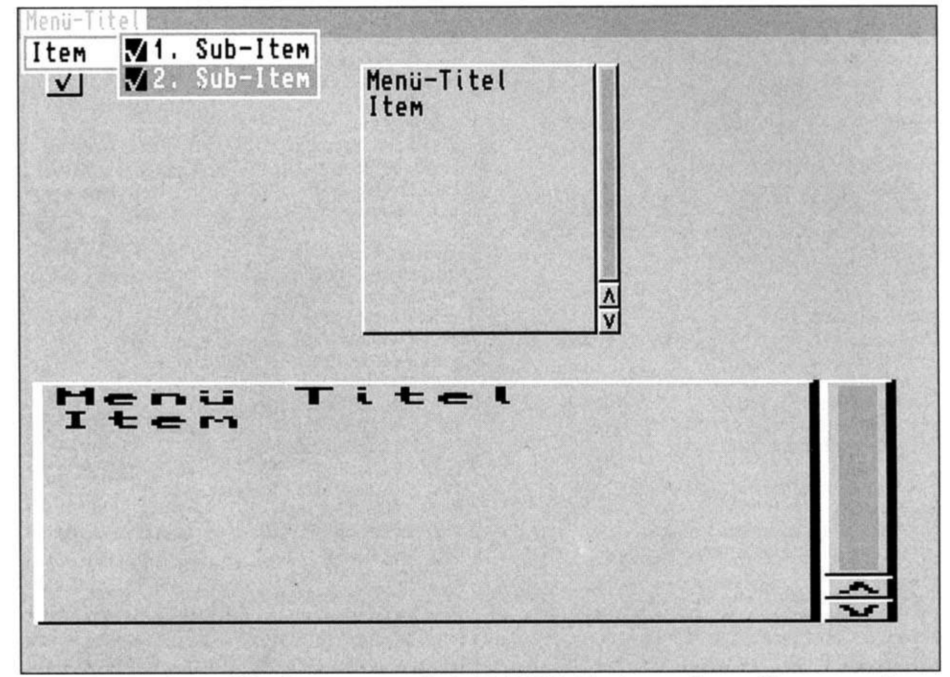
Alles Assembler

Die Handhabung der Systemfunktionen in Assembler ist komplizierter als z.B. in BASIC oder C. Aber dafür erhält man superschnelle Programme. Hier zeigen wir Ihnen, wie man in Assembler mit Fenstern und Menüs etc. unter OS 2.0 jongliert.

von Markus Adamski

An dieser Stelle präsentieren wir Ihnen ein Assemblerlisting, das den Umgang mit dem AMIGA-OS2.0 in bezug auf Screens, Windows und einige weitere Features für Assembler-Programmierer verdeutlicht. Das Programm dient als Ergänzung des Assemblerkurses im AMIGA-Magazin (Ausgabe 6/92 bis 1/93), der sich ausschließlich mit System 1.3 beschäftigte.

Das Source-Listing erklärt sich weitgehend selbst. Das Programm wird durch Druck einer beliebigen Taste beendet. Das Bild rechts zeigt, was herauskommt, wenn Sie das Programm auf Ihrem Amiga laufen lassen. ■



Alles da: In Assembler kann man wie in BASIC, C oder anderen Programmiersprachen Screens mit Menüs und Gadgets programmieren

```
*************************************************
*  Tips zu OS2.0:                               *
*  + ListView-Gadget incl. Listenverwaltung     *
*  + CheckBox-Gadget                            *
*  + BitMapScale()                              *
*  + Screen & Windows                           *
*  + Menüs                                      *
*  von Markus Adamski                           *
* Assembler: Devpac 2.14D                       *
*************************************************

  opt c+,ow+,o1+,o3+,o4+,o5+,s-

 incdir "DEVPAC:Includes/"
 include "exec/exec.i"
 include "graphics/displayinfo.i"
 include "graphics/scale.i"
 include "intuition/screens.i"
 include "libraries/gadtools.i"
 include "lvo.i"

 STRUCTURE Internal,0
 APTR _IntBase ; Libraries-Basis-Adressen
 APTR _GadBase
 APTR _GfxBase
 APTR ScrHandle ; -> Screen-Struktur
 APTR WinHandle ; -> Windows-Struktur
 APTR WinRPort ; -> Rastport des Windows
 APTR LV_Gad  ; -> ListView-Gadget
 APTR GetVisual ; Ergebnis von GetVisualInfoA()
 APTR MenuAddr ; Ergebnis von CreateMenusA()
 APTR GadgetList ; -> Gadget-Liste
 UBYTE Flags
 LABEL Int_SIZEOF

CALL  MACRO
 IFEQ NARG-2  ; 2 Werte übergeben ?
  IFND \2  ;  Ja -> 2. Wert definiert ?
   move.l 4.W,a6  ;   Nein -> ExecBase nach a6
  ELSEIF
   move.l \2(a5),a6 ;   Sonst Basis auslesen
  ENDC   ; Bei Übergabe nur eines Wertes
 ENDC   ;  wird a6 nicht verändert !
 jsr _LVO\1(a6) ; Aufruf
 ENDM

OPENLIB  MACRO
 lea \1(pc),a1
 moveq #\2,d0
 CALL OpenLibrary,EXEC
 move.l d0,\3(a5)
 ENDM

MENU  MACRO
 dc.b \1,0
 dc.l \2,0
 dc.w \3
 dc.l 0,0
 ENDM

TRUE = -1
FALSE = 0
FirstEntry = 0

 lea StrucPtr(pc),a5 ;Zeiger auf eigene Struktur

;*** Libraries öffnen
 OPENLIB IntName,37,_IntBase ; Libraries öffnen
 beq WrongOS
 OPENLIB GadName,37,_GadBase
 beq WrongOS
 OPENLIB GfxName,37,_GfxBase
 beq WrongOS

;*** Leere Liste erzeugen
 lea ListStruktur(pc),a0 ; Leere Liste erzeugen
 move.l a0,LH_TAILPRED(a0) ;LH_TAILPRED->LH_HEAD
 move.l a0,LH_HEAD(a0)  ; LH_HEAD -> LH_TAIL
 addq.l #4,LH_HEAD(a0)
 clr.l LH_TAIL(a0)  ; LH_TAIL = 0

;*** Screen öffnen
 lea ScrTags(pc),a1  ; Screen öffnen
 sub.l a0,a0
 CALL OpenScreenTagList,_IntBase
 move.l d0,ScrHandle(a5) ; Handle merken
 beq NoScreen

;*** Window öffnen
 lea WinTags(pc),a1  ; Window öffnen
 move.l d0,ti_Data(a1) ;Zeiger->Screen eintragen
 sub.l a0,a0
 CALL OpenWindowTagList
 move.l d0,WinHandle(a5) ; Handle merken
 beq NoWin
 move.l d0,a0   ; Rastport-Adresse holen
 move.l wd_RPort(a0),a0
 move.l a0,WinRPort(a5)  ;  und merken
;*** VisualInfo-Struktur erzeugen
 move.l ScrHandle(a5),a0 ; VisualInfo-Struktur
 lea VInfo_Tag(pc),a1 ;  erzeugen
 CALL GetVisualInfoA,_GadBase
 move.l d0,GetVisual(a5)
;*** Vorbereitung für NewGadgets
 lea GadgetList(a5),a0 ;Datenbereich reservieren
 CALL CreateContext  ; d0 = Zeiger
;*** ListView-Gadget erzeugen
 lea List_Gad_Tag(pc),a2 ; ListView-Gadget
 lea ListViewStruc(pc),a1
 move.l GetVisual(a5),gng_VisualInfo(a1)
 move.l d0,a0   ; Erg. von CreateContext()
 moveq #LISTVIEW_KIND,d0
 CALL CreateGadgetA
 move.l d0,LV_Gad(a5) ; Zeiger auf Gadget merken
```

»Intuition_Bsp.asm«: Menüs, Gadgets, Screens usw. in Assembler (Anfang)

```
;*** CheckBoxGadget erzeugen
 lea Check_Gad_Tag(pc),a2 ; CheckBox-Gadget
 lea CheckGadStruc(pc),a1
 move.l GetVisual(a5),gng_VisualInfo(a1)
 move.l d0,a0   ; Erg. von CreateGadgetA()
 moveq #CHECKBOX_KIND,d0
 CALL CreateGadgetA
;*** Menü erstellen und an Window übergeben
 lea MenuStruct(pc),a0 ; Menü erstellen
 lea MenuTag(pc),a1
 CALL CreateMenusA
 move.l d0,MenuAddr(a5)  ; Menü-Adresse merken
 move.l d0,a0
 move.l GetVisual(a5),a1
 lea LayoutTag(pc),a2
 CALL LayoutMenusA
 move.l WinHandle(a5),a0 ; Menü einhängen
 move.l MenuAddr(a5),a1
 CALL SetMenuStrip,_IntBase
;*** Gadgets zeichnen
 move.l WinHandle(a5),a0 ; Gadget in Window-
 move.l GadgetList(a5),a1 ;  Struktur eintragen
 moveq #-1,d0
 moveq #-1,d1
 sub.l a2,a2
 CALL AddGList
 move.l GadgetList(a5),a0 ; Gadgets zeichnen
 move.l WinHandle(a5),a1
 sub.l a2,a2
 moveq #-1,d0
 CALL RefreshGList
 move.l WinHandle(a5),a0
 sub.l a1,a1
 CALL GT_RefreshWindow,_GadBase

; *** Liste mit Einträgen erzeugen
; *** AB HIER DARF A0 NICHT VERÄNDERT WERDEN !!!

 lea ListStruktur(pc),a0 ; Listenkopf initialisieren
 move.l a0,LH_HEAD(a0)
 add.l #LH_SIZE,(a0)  ; LH_HEAD -> 1. Node
 clr.l LH_TAIL(a0)
 clr.l LH_TAILPRED(a0)
 clr.w LH_TYPE(a0)
 lea LH_SIZE(a0),a0
 bset #FirstEntry,Flags(a5) ; Erster Eintrag !
 lea MenuTxt_1(pc),a1 ; Namen in Liste eintragen
 bsr AddList
 lea MenuTxt_2(pc),a1
 bsr AddList
 lea ListStruktur(pc),a1 ; Leeren Eintrag
                          ;an Listenende
 move.l a1,LN_SUCC(a0)  ; LN_SUCC -> LH_TAIL
 addq.l #LH_TAIL,(a0)
 move.l a0,LH_TAILPRED(a1) ; LH_TAILPRED -> Node
 move.l a0,LN_PRED(a0)  ; LN_PRED -> LN_SUCC (!)
 sub.l #LN_SIZE+4,4(a0)
 clr.l LN_NAME(a0)
 ; * BIS HIER DARF A0 NICHT VERÄNDERT WERDEN !!!
;*** ListView-Gadget neu zeichnen
 move.l LV_Gad(a5),a0  ; List an ListView-
 move.l WinHandle(a5),a1 ;  Gadget übergeben
 sub.l a2,a2
 lea List_Gad_Tag(pc),a3
 CALL GT_SetGadgetAttrsA

;*** BitMap-Bereich vergrößern
 move.l ScrHandle(a5),a1 ;Struktur initialis.,
 lea sc_BitMap(a1),a1 ;  in der die Werte zur
 lea ScaleArgs(pc),a0 ;  Vergrößerung eines
 move.l a1,bsa_SrcBitMap(a0) ;BitPlane-Bereichen
 move.l a1,bsa_DestBitMap(a0) ; benötigt
 move.w #240,bsa_SrcX(a0) ;Quell-Bereich links/oben
 move.w #20,bsa_SrcY(a0)
 move.w #180,bsa_SrcWidth(a0) ; Quell-Ausdehnung
 move.w #100,bsa_SrcHeight(a0)
 move.w #10,bsa_DestX(a0) ; Ziel-Bereich links oben
 move.w #130,bsa_DestY(a0)
 move.w #180,bsa_XSrcFactor(a0) ;Vergrößerung als
 move.w #67,bsa_YSrcFactor(a0)  ;Bruch darstellbar:
 move.w #600,bsa_XDestFactor(a0) ;Zähler als Ziel-,
 move.w #60,bsa_YDestFactor(a0) ;Nenner als Quell-
 move.w #600,bsa_DestWidth(a0) ; Faktor eingetragen
 move.w #89,bsa_DestHeight(a0)  ; X: 600/180 = 3.33
 CALL BitMapScale,_GfxBase  ; Y: 60/67   = 0.89
```

```
;*** Eingabe aus UserMsgPort des Windows lesen
Eingabe_holen:
 move.l WinHandle(a5),a3    ; UserPort-Adresse des
 move.l wd_UserPort(a3),a3  ; Windows holen
Message_holen: move.l a3,a0 ; Message abholen
 CALL GT_GetIMsg,_GadBase
 tst.l d0                   ; War eine da ?
 bne.s Auswerten            ; Ja -> auswerten
 moveq #-1,d0               ; Nein -> Signalbit holen
 CALL AllocSignal,EXEC
 tst.b d0                   ; War eins frei ?
 bmi.s Message_holen        ; Nein -> Message holen
 move.b d0,MP_SIGBIT(a3) ; Bit in MessagePort
 move.l ThisTask(a6),MP_SIGTASK(a3) ; Task-Adresse
 clr.b MP_FLAGS(a3)         ; Keine Flags
 move.l a3,a0               ; Auf Signal warten
 CALL WaitPort
 move.b #PA_IGNORE,MP_FLAGS(a3) ; Nicht reagieren
 move.b MP_SIGBIT(a3),d0         ; Signalbit holen,
 CALL FreeSignal                 ;  freigeben,
 bra.s Message_holen        ;  und Nachricht holen

Auswerten: move.l d0,a1
 CALL GT_ReplyIMsg,_GadBase   ; Beantwortem
;*** Gadgets entfernen
 move.l GetVisual(a5),a0      ; Gadgets entfernen
 CALL FreeVisualInfo,_GadBase
 move.l GadgetList(a5),a0
 CALL FreeGadgets
;*** Menüs entfernen
 move.l MenuAddr(a5),a0       ; Menüs freigeben
 CALL FreeMenus,_GadBase
 move.l WinHandle(a5),a0      ; Menü entfernen
 CALL ClearMenuStrip,_IntBase
;*** Fenster schließen
 move.l WinHandle(a5),a0      ; Fenster schließen
 CALL CloseWindow
;*** Screen schließen
NoWin:  move.l ScrHandle(a5),a0
 CALL CloseScreen             ; Screen schließen
;*** Libraries schließen
NoScreen: move.l _IntBase(a5),a1 ; Libraries zu
 CALL CloseLibrary,EXEC
 move.l _GadBase(a5),a1
 CALL CloseLibrary
 move.l _GfxBase(a5),a1
 CALL CloseLibrary

WrongOS: moveq #0,d0   ; ENDE
 rts
*** Name in Liste einhängen ***
*** a0 = Zeiger auf freien Eintrag
*** a1 = Zeiger auf String

AddList: btst #FirstEntry,Flags(a5) ; 1. Eintrag ?
 bne.s 1$   ; Ja -> Spezialfall
 move.l a0,LN_SUCC(a0)
 add.l #LN_SIZE,(a0)  ; LN_SUCC -> Nächste Node
 move.l a0,LN_PRED(a0)  ; LN_PRED -> LN_SUCC (!)
 sub.l #LN_SIZE+4,LN_PRED(a0) ;Zeiger um Länge einer
 clr.w LN_TYPE(a0)             ; Node auf Vorgänger
 move.l a1,LN_NAME(a0)         ; verkürzen !!!
 lea LN_SIZE(a0),a0
 rts

1$:  bclr #FirstEntry,Flags(a5)
 move.l a0,LN_SUCC(a0)
 add.l #LN_SIZE,(a0)     ; LN_SUCC -> Nächste Node
 move.l a0,LN_PRED(a0)   ; LN_PRED -> LH_HEAD (!)
 sub.l #LH_SIZE+4,LN_PRED(a0) ;Zeiger um Länge eines
 clr.w LN_TYPE(a0)             ; List-Kopfes auf den
 move.l a1,LN_NAME(a0)         ;  Vorgänger verkürzen!
 lea LN_SIZE(a0),a0
 rts

StrucPtr: dcb.w (Int_SIZEOF+1)/2,0
IntName: dc.b "intuition.library",0
GadName: dc.b "gadtools.library",0
GfxName: dc.b "graphics.library",0

ScrTitle: dc.b "Testfenster",0

MenuTxt_1: dc.b "Menü-Titel",0
MenuTxt_2: dc.b "Item",0
MenuTxt_3: dc.b "1. Sub-Item",0
```

```
MenuTxt_4: dc.b "2. Sub-Item",0
 even

WinTags: dc.l WA_CustomScreen,0
 dc.l WA_Left,0
 dc.l WA_Top,0
 dc.l WA_Width,640
 dc.l WA_Height,256
 dc.l WA_Backdrop,TRUE
 dc.l WA_Borderless,TRUE
 dc.l WA_IDCMP,IDCMP_VANILLAKEY
 dc.l TAG_DONE

ScrTags: dc.l SA_Title,ScrTitle
 dc.l SA_Colors,ColorTab
 dc.l SA_Pens,Pens_3D_Struct
 dc.l SA_DisplayID,PAL_MONITOR_ID|HIRES_KEY
 dc.l SA_Width,640
 dc.l SA_Height,256
 dc.l SA_Depth,3
 dc.l SA_Type,CUSTOMSCREEN
 dc.l SA_DetailPen,7
 dc.l SA_BlockPen,3
 dc.l TAG_DONE

Pens_3D_Struct: dc.w 0,0,1,7,1,3,1,-1

ColorTab: dc.w 0,10,10,10
 dc.w 1,0,0,1
 dc.w 2,15,15,15
 dc.w 3,6,8,11
 dc.w 4,15,15,15
 dc.w 5,0,0,1
 dc.w 6,0,0,0
 dc.w 7,14,14,14
 dc.w -1

ListViewStruc:
 dc.w 240,20,180,100  ; x, y, Breite, Höhe
 dc.l 0,Font   ; Label, TextAttr
 dc.w 10   ; GadgetID
 dc.l PLACETEXT_IN,0,0
List_Gad_Tag:
 dc.l GTLV_Labels,ListStruktur ; Tag für ListView
 dc.l GA_Disabled,FALSE
 dc.l GTLV_ReadOnly,FALSE
 dc.l TAG_DONE
ListStruktur:
 dcb.b LH_SIZE+LN_SIZE*17,0 ; Liste für ListView

CheckGadStruc:
 dc.w 20,20,140,20  ; CheckBox-Gadget
 dc.l 0,Font
 dc.w 11
 dc.l PLACETEXT_IN,0,0

Check_Gad_Tag:
 dc.l GTCB_Checked,TRUE
 dc.l TAG_DONE

MenuStruct:
 MENU NM_TITLE,MenuTxt_1,ITEMTEXT
 MENU NM_ITEM,MenuTxt_2,ITEMTEXT
 MENU NM_SUB,MenuTxt_3,CHECKED!CHECKIT!MENUTOGGLE
 MENU NM_SUB,MenuTxt_4,CHECKED!CHECKIT!MENUTOGGLE
 MENU NM_END,0,0
MenuTag:
 dc.l GTMN_FrontPen,2  ; Tags für Menüs
 dc.l GTMN_FullMenu,TRUE
 dc.l TAG_DONE
LayoutTag: dc.l GTMN_TextAttr,Font
 dc.l TAG_DONE

VInfo_Tag: dc.l TAG_DONE  ; Tag für GetVisualInfoA()

Font:  dc.l FontName  ; TextAttr-Struktur
 dc.w 9,0
FontName: dc.b "topaz.font",0
 even

ScaleArgs: dcb.w 12,0
 dc.l 6,0
```


»Intuition_Bsp.asm«: Menüs, Gadgets, Screens usw. – alles in Assembler

BASIC mit System

Freie Auswahl

Mit Hilfe der»asl.library« können Sie per BASIC-Programm Font- oder Dateiauswahlfenster programmieren, die sog. Requester. Wir zeigen Ihnen Beispiele für die gebräuchlichsten Requesterformen.

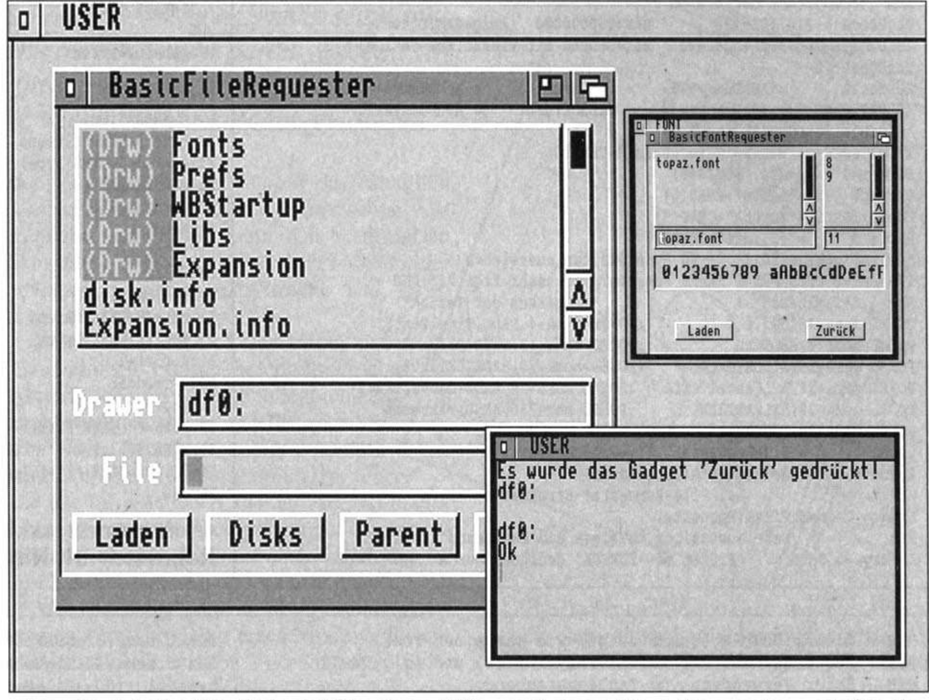
von Ralf Schmidt

Seit OS 2.0 gibts im Amigabetriebssystem standardmäßig die »asl.library« mit vielen leistungsfähigen Funktionen für Requester und Fontverwaltung. Hier nun ein paar Beispiele, wie Sie die Bibliothek von BASIC aus nutzen. Voraussetzung für den Einsatz ist die »asl.library« im logischen Laufwerk »LIBS:« und Kickstart 2.0!

Außerdem brauchen Sie für BASIC die »bmap«-Datei der »asl.library« im Verzeichnis »LIBS:«. Sie finden diese Datei u.a. auf der PD-Diskette zu diesem Heft; Sie können die Datei aber auch mit dem Listing »MakeAslBmap« auf der nächsten Seite selbst erzeugen. Das Programm erstellt die Datei »asl.bmap« im logischen Laufwerk »LIBS:«, mit deren Hilfe AmigaBASIC die Register-Aufrufkonventionen zu jeder Routine in der »asl.library« erhält.

Hier nun die Programme, die zeigen wie Sie einige Requester der »asl.library« nutzen:

❒ »FileRequest.bas« (unten) demonstriert, wie man einen AslFileRequester in AmigaBASIC und dessen Auswertung realisiert (Bild).



»Komfortabel«: Ein Beispiel für einen File- und einen Font-Requester, wie Sie ihn mit der »asl.library« (ab 2.0) selbst programmieren können – in BASIC!

❒ Das nächste Listing »UserFileRequest.bas« auf der folgenden Seite zeigt unter Verwendung von sog. TagItems wie man einen File-Requester wie den des ersten Beispiels an eigene Bedürfnisse anpaßt. Versuchen Sie durch Ändern der einzelnen Parameter das Ergebnis zu ändern und Ihren eigenen Wünschen anzupassen.

❒ Das letzte Programm »FontRequest.bas« öffnet einen Requester zur Auswahl von Fonts aus dem logischen Laufwerk »FONTS:«. So können Sie z.B. Schrift für einen Text wählen. ■

```
REM - Das Programm zeigt, wie man ein
REM - File-Requester mit der ASL.Library (OS2.0)
REM - in AmigaBASIC realisiert.
REM - »asl.bmap« in LIBS: erforderlich

  LIBRARY "asl.library"

DECLARE FUNCTION AllocFileRequest& LIBRARY
DECLARE FUNCTION RequestFile& LIBRARY
DECLARE FUNCTION FreeFileRequest& LIBRARY

Request&=AllocFileRequest&  'Asl File-Requester-Struktur
Erg&=RequestFile&(Request&) 'Auswahl zeigen, Bdg auswerten

IF Erg&=0 THEN      'Prüfen ob 'CANCEL' gedrückt wurde
 PRINT "Es wurde das Gadget 'CANCEL' gedrückt!"
END IF

File&=PEEKL(Request&+4) 'Adresse des File-Namens herausfinden
Dir&=PEEKL(Request&+8)  'Adresse des dir(pfad)-Namens finden

GOSUB eingabeauswerten 'File- und Pfadname in Strings wandeln

fehler&=FreeFileRequest&(Request&) 'Struktur freigeben

PRINT Dir$         'Name des Pfades
PRINT File$        'Name des Files
PRINT Pfadfile$    'beide zusammen

  LIBRARY CLOSE
END
eingabeauswerten:
File$=""
readfile:
byte=PEEK(File&)
IF byte=0 THEN fileend
File$=File$+CHR$(byte)
File&=File&+1
GOTO readfile

fileend:
Dir$=""
readdir:
byte=PEEK(Dir&)
IF byte=0 THEN dirend
Dir$=Dir$+CHR$(byte)
Dir&=Dir&+1
GOTO readdir

dirend:
IF Dir$="" THEN Pfadfile$=File$:RETURN
IF RIGHT$(Dir$,1)=":"THEN Pfadfile$=Dir$+File$:RETURN
Pfadfile$=Dir$+"/"+File$
RETURN          '  © 1993 M&T
```

»Filerequest_asl.bas«: AmigaBASIC nutzt in diesem Beispiel die »ASL.library«, um einen Filerequester darzustellen

```
REM - Dieses Demo-Programm zeigt, wie man einen File-Requester
REM - mit der asl.library (OS2.0) und AmigaBASIC unter
REM - Verwendung von TagItems an eigene Bedürfnisse anpaßt.
 LIBRARY "asl.library"
 LIBRARY "exec.library"
DECLARE FUNCTION AllocAslRequest& LIBRARY
DECLARE FUNCTION RequestFile& LIBRARY
DECLARE FUNCTION FreeFileRequest& LIBRARY
DECLARE FUNCTION AllocMem& LIBRARY
DECLARE FUNCTION FreeMem& LIBRARY
REM  ** TagItems definieren **
TagUser& =-2147483648#     ' HEX=$80000000  (AmigaBASIC kennt
ASLDummy&=TagUser&+524288& ' HEX=$80000 nur vierst.Hex-Zahlen)
TagDone& =0
ASLHail&       = ASLDummy&+1    :  ASLDir&        = ASLDummy&+9
ASLPATTERN&    = ASLDummy&+10   :  ASLOKText&     = ASLDummy&+18
ASLCancelText&= ASLDummy&+19
TitelText$  ="BasicFileRequester"+CHR$(0)
DirName$    ="df0:"+CHR$(0)
OKText$     ="Laden"+CHR$(0)
CancelText$ ="Zurück"+CHR$(0)
Patter$     ="#?.info"+CHR$(0)
mem&=AllocMem&(50,1)  ' 50 Byte dürften ausreichen
IF mem&=0 THEN PRINT "Kein Speicherplatz mehr frei ?!":END
REM   **TagItems**                    **Adressen der Texte**
POKEL mem&,ASLHail&              :POKEL mem&+4,SADD(TitelText$)
POKEL mem&+8,ASLDir&             :POKEL mem&+12,SADD(DirName$)
POKEL mem&+16,ASLOKText&         :POKEL mem&+20,SADD(OKText$)
POKEL mem&+24,ASLCancelText&     :POKEL mem&+28,SADD(CancelText$)
POKEL mem&+32,ASLPATTERN&        :POKEL mem&+36,SADD(Patter$)
POKEL mem&+40,TagDone&
type&=0  '0 = FileRequester, 1 = FontRequester
Request&=AllocAslRequest&(type&,mem&)
REM                Asl File-Requester-Struktur besorgen
Erg& = RequestFile&(Request&)
REM              Dateiauswahlbox anzeigen und Bedienung auswerten
IF Erg&=0 THEN       'Prüfen ob 'ZURÜCK' gedrückt wurde
 PRINT "Es wurde das Gadget 'ZurÜck' gedrückt!"
END IF
File&=PEEKL(Request&+4) 'Adresse des file-namens herausfinden
Dir&=PEEKL(Request&+8)  'Adresse des dir(pfad)-namens finden
GOSUB eingabeauswerten 'File- und Pfadname in Strings wandeln
fehler&=FreeFileRequest&(Request&) 'Requester-Struktur freigeben
PRINT Dir$         'Name des Pfades
PRINT File$        'Name des Files
PRINT Pfadfile$   'beide zusammen
guru&=FreeMem&(mem&,50)
  LIBRARY CLOSE
END
eingabeauswerten:
File$=""
readfile:
byte=PEEK(File&)
IF byte=0 THEN fileend
File$=File$+CHR$(byte)
File&=File&+1
GOTO readfile
fileend:
Dir$=""
readdir:
byte=PEEK(Dir&)
IF byte=0 THEN dirend
Dir$=Dir$+CHR$(byte)
Dir&=Dir&+1
GOTO readdir
dirend:
IF Dir$="" THEN Pfadfile$=File$:RETURN
IF RIGHT$(Dir$,1)=":"THEN Pfadfile$=Dir$+File$:RETURN
Pfadfile$=Dir$+"/"+File$
RETURN
```


»Fontrequest_asl.bas«: So programmiert man einen Font-Requester mit Hilfe der »asl.library«

```
REM - Dieses Demo-Programm zeigt, wie man einen Font-
REM - Requester mit der asl.library (OS2.0) und AmigaBASIC
REM - unter Verwendung von TagItems anwendet.
 LIBRARY "asl.library"
 LIBRARY "exec.library"
DECLARE FUNCTION AllocAslRequest& LIBRARY
DECLARE FUNCTION RequestFile& LIBRARY
DECLARE FUNCTION FreeFileRequest& LIBRARY
DECLARE FUNCTION AllocMem& LIBRARY
DECLARE FUNCTION FreeMem& LIBRARY
REM  ** TagItems definieren **
TagUser& =-2147483648#    'HEX=$80000000 (AmigaBASIC kennt nur
ASLDummy&=TagUser&+524288& 'HEX=$80000 nur vierst. HEX-Zahlen)
TagDone& =0
ASLHail&       = ASLDummy&+1   :    ASLDir&        = ASLDummy&+9
ASLFontName&   = ASLDummy&+10  :    ASLFontHeight&= ASLDummy&+11
ASLOKText&     = ASLDummy&+18  :    ASLCancelText&= ASLDummy&+19
TitelText$  ="BasicFontRequester"+CHR$(0)
OKText$     ="Laden"+CHR$(0)
CancelText$ ="Zurück"+CHR$(0)
FontName$   ="topaz.font"+CHR$(0)     ' Vorgabe
Height&     =11  ' Vorgabe
mem&=AllocMem&(50,1)  ' 50 Byte dürften ausreichen
IF mem&=0 THEN PRINT "Kein Speicherplatz mehr frei ?!":END
REM    **TagItems**                  **Adressen der Texte**
POKEL mem&,ASLHail&             :POKEL mem&+4,SADD(TitelText$)
POKEL mem&+8,ASLOKText&         :POKEL mem&+12,SADD(OKText$)
POKEL mem&+16,ASLCancelText&    :POKEL mem&+20,SADD(CancelText$)
POKEL mem&+24,ASLFontName&      :POKEL mem&+28,SADD(FontName$)
POKEL mem&+32,ASLFontHeight&   :POKEL mem&+36,Height&
POKEL mem&+40,TagDone&
type&=1  '0 = FileRequester, 1 = FontRequester
Request&=AllocAslRequest&(type&,mem&) 'Asl Font-Req.-Struktur
Erg&=RequestFile&(Request&) 'Fontauswahl zeigen, Bdg auswerten
IF Erg&=0 THEN      'Prüfen ob 'ZURÜCK' gedrückt wurde
 PRINT "Es wurde das Gadget 'ZurÜck' gedrückt!"
END IF
Font&=PEEKL(Request&+8)  'Adresse des Font-Namens herausfinden
Height&=PEEKW(Request&+12)  'FontHöhe herausfinden
GOSUB eingabeauswerten 'FontName in String wandeln
fehler&=FreeFileRequest&(Request&) 'Struktur freigeben
PRINT "Fonthöhe  "Height&
PRINT "Fontname   "Font$
guru&=FreeMem&(mem&,50)
  LIBRARY CLOSE
END
eingabeauswerten:
Font$=""
readfont:
byte=PEEK(Font&)
IF byte=0 THEN RETURN
Font$=Font$+CHR$(byte)
Font&=Font&+1
GOTO readfont        '
```


»Fontrequest_asl.bas«: So programmiert man einen Font-Requester mit Hilfe der »asl.library«

```
REM Erstellt die 'asl.bmap' Datei im logischen Laufwerk 'LIBS:'
OPEN "LIBS:asl.bmap" FOR OUTPUT AS #1
FOR t= 0 TO 55
 READ x$
 x$="&H"+x$        :    x%=VAL(x$)
 y&=y&+x%
 PRINT#1,MKI$(x%);
NEXT
CLOSE #1
KILL "LIBS:asl.bmap.info"
IF y&<\<>>1075711& THEN
 PRINT "In den Datazeilen liegt ein Tipfehler vor!!"
 ELSE
  PRINT "Die 'asl.bmap' befindet sich jetzt im logischen Laufwerk 'LIBS:'"
END IF
DATA  416C,6C6F,6346,696C,6552,6571,7565,7374,00FF,E200
DATA  4672,6565,4669,6C65,5265,7175,6573,7400,FFDC,0900
DATA  5265,7175,6573,7446,696C,6500,FFD6,0900,416C,6C6F
DATA  6341,736C,5265,7175,6573,7400,FFD0,0109,0046,7265
DATA  6541,736C,5265,7175,6573,7400,FFCA,0900,4173,6C52
DATA  6571,7565,7374,00FF,C409,0A00        ;
```


»Makeasl.bas«: Die »asl.bmap« ist Voraussetzung zur Nutzung der »asl.library« – basteln Sie die Datei ggf. selbst

Richtig dokumentieren

Des Programmierers Leid

Programme zu dokumentieren, ist etwas Schönes, wenn man es nicht selbst machen muß. Oft kommt man als Programmierer aber nicht daran vorbei, Programme zu beschreiben – und sei es nur für persönliche Informationen, damit man nach ein paar Wochen noch versteht, wie man eine Aufgabe gelöst hat.

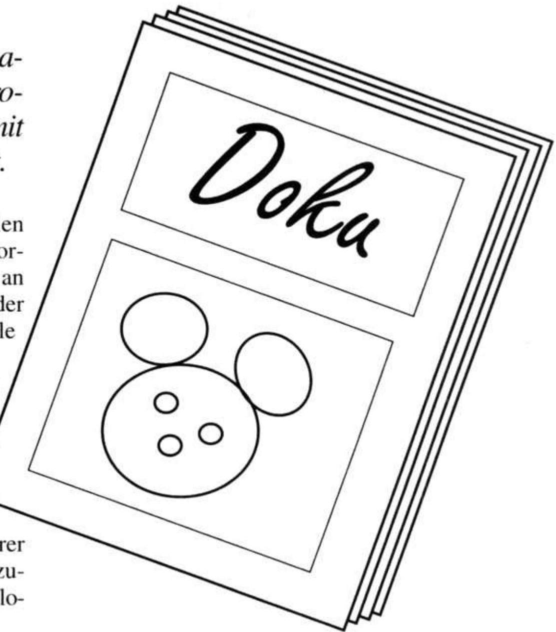
von Matthias Straß

Entwickelt man größere Programme, ist es unerläßlich, eine Dokumentation zu führen, um einen Überblick über deren Funktionen zu gewährleisten. Vielleicht hat man schon in zurückliegenden Projekten Unterprogramme geschrieben, die man später, bis auf kleine Änderungen, gut gebrauchen könnte – schade, wenn man dann nicht mehr weiß, wie der Algorithmus funktioniert.

Aber was gehört in eine Dokumentation? Meist macht man sich keine Gedanken und nimmt mal diesen, mal jenen Punkt auf. Wenn es drauf ankommt, stellt sich dann heraus, daß oft gerade das Entscheidende vergessen wurde. Nach Jahren Erfahrung möchte der Autor hier ein Konzept vorstellen, um Dokumentationen schnell und einheitlich zu gestalten:

Dokumentationsformulare:

Die zentrale Rolle spielen Formulare. Trotz ihres bürokratischen Beigeschmacks bieten sie den Vorteil, daß man sich auf das Ausfüllen konzentrieren kann, da die Felder schon vorgegeben sind – jeder kann sich bestimmt an Formulare erinnern, in denen er die Felder »Name«, »Vorname«, »Straße« und viele mehr ausfüllen mußte.



Was hat das mit Dokumentation zu tun? Ganz einfach: Für die Beschreibung eines Moduls oder einer Funktion legt man einmal Formulare an, die man für jedes Modul (jede Funktion) kopiert, ausfüllt und sammelt. In diesen Formularen stehen die alle Informationen, die der Programmierer benötigt, um sich in einem Modul zurechtzufinden (»Modul« im Sinne einer Sammlung logisch zusammenhängender Funktionen).

Wie angedeutet, sind es zwei Formulare: das eine zur Beschreibung eines Moduls, das andere als Dokumentation einer Funktion. Beim Verwenden der Formulare sparen Sie viel Tipparbeit, die sonst nötig wäre, um allein den Satz an Feldbezeichnern einzugeben. Hiermit sind Feldnamen wie »Funktionsname:«, »Datum:«, »Beschreibung:« usw. gemeint.

Ein Formular besitzt als weitere Eigenschaft, daß sämtliche Felder immer in derselben Reihenfolge angeordnet sind. Wir statten unser Formular also ein einziges Mal mit den wichtigsten Felder aus, sortieren sie so, daß das Wichtigste am Anfang steht und legen es als Datei ab. Die Felder selbst markieren wir sinniger Weise mit einem Fragezeichen. Lücken,

```
/* ##########################################################################
  MODUL......: ????.C
  EXPORT FNKT:
                                              ????.C - ?
  EXPORT VARS:
                                              ????.C - ?
----------------------------------------------------------------------------
  IMPORT FNKT: ?
  IMPORT VARS: ?
----------------------------------------------------------------------------
  WICHTIG....: ?
----------------------------------------------------------------------------
  DOKU.......: ?

  STATUS.....: < > Actual                            < > Released
               <X> Changing        < > Version update

  HIST/BUGS..: Vers/TT-MMM-JJ/Wer   Was
               0.00/?        /?     Ersterstellung
----------------------------------------------------------------------------
  AUTOR......: ?                    [?]
  COPYRIGHT..: (c) 1993

  UMGEBUNG...: ?

######################################################################### */

/****************************************************************** MAKE */
/*--------------------------------------------------------------- options */
/*
        ?
*/
/************************************************************** INCLUDES */
#define MOD_????
/*----------------------------------------------------------------- amiga */
#include <exec/types.h>
/*----------------------------------------------------------------- std_c */
/*---------------------------------------------------------------- module */

/************************************************************** TYPEDEFS */
/*------------------------------------------------------------- ? */
/*------------------------------------------------------------- ? */

/*************************************************************** DEFINES */
/*-------------------------------------------------------------- ? */
/*-------------------------------------------------------------- ? */

/**************************************************************** MACROS */
/*--------------------------------------------------------------- ? */
/*--------------------------------------------------------------- ? */

/**************************************************************** EXPORT */
/*------------------------------------------------------------- functions */
/*------------------------------------------------------------- variables */

/**************************************************************** IMPORT */
/*------------------------------------------------------------- functions */
/*------------------------------------------------------------- variables */

/**************************************************************** INTERN */
/*------------------------------------------------------------- functions */
/*------------------------------------------------------------- variables */

/* ##########################################################################
################################################################ IMPLEMENTATION
########################################################################## */

/*
########################################################### GRUPPE: ?
*/

/********************************************************** ??()

  RÜCKGABE...: ?
  BEDINGUNG..: ?
  --------------------------------------------------------------------------
  FUNKTIONEN.: ?
  VARIABLEN..: ?
  --------------------------------------------------------------------------
  DOKU.......: ?
  LÖSUNG.....: ?
  --------------------------------------------------------------------------
  HIST/BUGS..: Vers/TT-MMM-JJ/Wer   Was
               0.00/?        /?     Ersterstellung
----------------------------------------------------------------------------
*/
? ??              /* ? */
(  ?     /* ? */
  ,?     /* ? */
  ,?     /* ? */
)
{
} /*********************************************************** ??() */

/*############################################### end of module ???? */
```

Beispiel: Unser Formular für den Modulkopf, das Sie für jedes neue Programm einsetzen können

die sich beim ersten Ausfüllen zwangsläufig ergeben, können, wenn die fehlenden Informationen vorhanden sind, einfach erkannt werden – durch das »?« hinter dem Feldbezeichner.

Werfen wir einen Blick auf ein solches Formular – z.B. auf das der Modul-Dokumentation (siehe Abbildung). Hier wird, so hoffen wir, jedem klar, was mit »Feldbezeichner« bzw. mit den »Fragezeichen«, die das dazugehörige »Feld« markieren sollen, gemeint ist.

Mit Sicherheit ist Ihnen schon die eigenwillige Stellung und Schreibweise der einzelnen Überschriften (MAKE, options, INCLUDES, amiga, usw.) aufgefallen. Die Stellung am äußerst rechten Rand hat zum Grund, daß gerade die linke Seitenhälfte mit Text (nämlich dem Programm-Code) ziemlich ausgelastet ist. Also kann das Auge auf der weniger »beschriebenen« rechten Seite viel schneller Informationen identifizieren. Es ist z.B. wesentlich leichter, den so markierten Beginn und Abschluß einer Funktion auszumachen – probieren Sie es aus! Aus dieser Positionierung folgt auch, daß man auf großartige Rahmungen (die üblichen Sternchenkästen um Überschriften) verzichten kann, denn die Rahmen sind einzig und allein dazu da, dem Auge einen Blickfang für die wichtigen Punkte zu verschaffen. An sonsten kosten sie nur Mühe und Platz.

Die Strichstärke (sprich: Zeilen aus »###«, »***«, »===« oder »---«) und die Groß- und Kleinschreibung sollen unterschiedliche Hierarchien der Überschriften verdeutlichen (dem Auge zur Hilfe). Das ganze Verfahren mag dem einen oder anderen etwas fremd vorkommen – macht nichts, probieren Sie es einfach einmal aus. Wir garantieren Ihnen, daß Sie von mal zu mal besser mit dem Dokumentieren vorankommen und sich, auch nach einer ganzen Weile, noch gut und schnell in Ihren Programmen zurechtfinden.

Der einzige Pferdefuß bei der Sache ist noch, daß die Zeilen noch von Hand so »nachjustiert« werden müssen, daß sie schön bündig am rechten Rand (Spalte 79) stehen – es sei denn, Ihr Editor unterstützt Sie mit umfangreichen Makrofunktionen.

Die Formulare können leicht den einzelnen Sprachen und natürlich Ihren individuellen Bedürfnissen angepaßt werden. Das einzige, was man wissen muß, ist, welche Zeichen in der betreffenden Sprache einen Kommentar öffnen und schließen. Denn diese Formulare sollen als Kommentar mit im Quell-Code stehen.

Einige Sprachen bieten eine komfortable Lösung an, die lediglich ein öffnendes und ein schließendes Kommentarzeichen erfordert, z.B.
– C oder AREXX: /* und */
– PASCAL: { und }

Die folgenden Sprachen besitzen lediglich einen Kommentarbeginn, weswegen hier jede Zeile des Formulars mit einem Kommentarzeichen eingeleitet werden muß; z.B. für
– BASIC: REM bzw. '
– Assembler: * bzw. ;

Die Formulare werden fix und fertig in einer Datei gespeichert. Feldinhalte, die sich nie ändern, z.B. Ihr Name als Autor, sollten gleich mit gespeichert werden.

Verwendung:
Um das Ausfüllen der Formulare möglichst effektiv zu gestalten, sollte man sich von seinem Editor kräftig helfen lassen (Makros), z.B. beim Hereinkopieren des Formulars. Es können, bequemerweise, Fragezeichengruppen gebildet werden, die als Platzhalter an den verschiedensten Stellen in den Formularen stehen, aber immer mit demselben Text ersetzt werden sollen (siehe »????« für den Modulname oder »??« für den Namen der Funktion). Sie sollten tunlichst zuerst die langen Gruppen ersetzen...

Die restlichen Fragezeichen springt man mit der Suchfunktion an und ersetzt sie durch die entsprechenden Informationen.

Ausfüllen automatisieren

Wer beim Betrachten des Modulformulars Felder mit den Bezeichnungen »PROGRAMM-NAME«, »PROJEKT«, usw. vermißt, sollte sich bewußt sein, daß diese in einem MODUL nichts zu suchen haben – Module sollen schließlich unabhängig von irgendwelchen Programmen sein!

Versions-Nummern:
Bevor wir beginnen, die einzelnen Felder zu erklären, reißen wir kurz das Thema Versions-Nummern an. Wir finden es auf jeden Fall sinnvoll, Versions-Nummern gezielt zu vergeben. Man kann sie nicht nur für komplette Programme vergeben (Ed 2.00), sondern auch für Texte, Module und Funktionen. Durch Versions-Nummern wird eine gewisse Fortentwicklung zum Ausdruck gebracht. Wir schlagen für die Vergabe von Versionsnummernfolgende Definition vor:

Eine Versions-Nummer besteht aus drei Ziffern: »X.YZ« (z.B. »1.15«):

❒ »X«: Die erste Ziffer sagt etwas über die Fortentwicklung aus. Befindet sich z.B. ein Modul im Entwicklungsstadium, sollte hier eine »0« vergeben werden. Ist es soweit, daß man das Modul anderen zur Verfügung stellen kann, d.h., daß es ausgetestet wurde und stabil läuft, vergibt man die Version »1.00«. Hat sich einiges durch Erweiterungen, Verbesserungen und Fehlerbeseitigungen verändert, erfolgt die Freigabe als Version »2.00«, usw. Dieses Setzen der Versions-Nummer wird im Modul mit dem Zustand »Versions-Update« gekennzeichnet (siehe Feld »ZUSTAND«).

❒ »Y«: Die Ziffer wird beim Beseitigen mittelschwerer Fehler oder beim Einführen effizienterer Algorithmen hochgezählt, automatisch setzt man die letzte Ziffer »Z« auf »0«.

❒ »Z«: Hochzählen bei Korrektur kleinerer Fehler oder bei »kosmetischen« Änderungen.

Die Felder des Modulkopfs:
Betrachten wir aber nun einmal die Felder des Modulformulars:

❒ `MODUL:      (Modulname)`
Aussagekräftiger Name der Datei, in welcher das Modul steckt.

❒ `EXPORT FNKT: (exportierte Funktionen)`
`EXPORT VARS: (exportierte Variablen)`
Funktionen bzw. Variablen in diesem Modul, »die ich nach außen weitergebe« (exportiere). Diese stehen anderen Modulen zur Benutzung offen; das Fragezeichen im Beispiellisting vor dem Bindestrich ist durch den Namen dieses Moduls, das zweite durch den Funktionsnamen bzw. durch Typ + Name der Variable zu ersetzen.

Weswegen hier Modul- und Funktionsname aufgeführt werden, hat einen praktischen Grund: Zum Beispiel arbeiten Sie gerade an einem Modul namens »EinAusgabe« und schreiben die Funktion »SortInput«. Dazu benötigen Sie die Funktion »SortArray()«, die im Modul »Sort« steht, d.h., die durch dieses Modul exportiert wird.

Also holen Sie sich das Modul »Sort« in den Editor und kopieren den unter »EXPORT FNKT« stehenden Eintrag »SORT.C – SortArray()» in Ihr aktuelles Modul. Natürlich nicht irgendwo hin, sondern an diejenigen Stellen des Modul- und Funktionskopfes, die mit »IMPORT FNKT« (siehe nächster Punkt) gekennzeichnet sind.

Der vorangestellte Modulname verweist auf die Quelle der Funktion, wo ich nähere Informationen finden kann. Leider hat die Sache einen Haken: Trotz aller Logik und Umsicht kann es vorkommen, daß ich Module zerlegen muß, vielleicht schon einfach deswegen, weil sie zu groß geworden sind – man geht von ca. 1000 Zeilen pro Modul aus.

Was macht man jetzt mit meinen schönen Verweisen auf die Funktions-Quellen?

Man könnte sämtliche Module nach dem Namen des zerlegten Moduls durchsuchen und für die betroffenen Funktionen die Quellenangabe ändern (Replace-Funktion). Zu beachten ist, daß auch die eine, alte »#include«-Direktive durch neue Anweisungen zu ersetzen ist – für jedes neue Modul eine. Diese Lösung ist jedoch nur praktikabel, wenn es sich um relativ wenige Module handelt.

Eine weitere, nicht so aufwendige Möglichkeit bietet sich, wenn man die neu entstandenen Module ins ursprüngliche Modul durch »#include«-Direktiven einbindet. Als Inhaltsverzeichnis (für die, vom Aufspalten betroffenen Module) kopieren Sie den hier beschriebenen Teil des Modulkopfs (EXPORT FNKT/VARS) der entstandenen Module unter die jeweilige »#include«-Anweisungen. So erhalten Sie die logische Einheit des ursprünglichen Moduls. Vorteil dieser Methode ist, daß sämtliche »#include«-Direktiven in alten Modulen bestehen bleiben können.

❒ Analoges wie für die beschriebenen Importe gilt für exportierte Variablen:
`IMPORT FNKT:   (importierte Funktionen)`
`IMPORT VARS:   (importierte Variablen)`
Aus welchen anderen Modulen verwendet dieses Modul welche Funktionen oder Variablen (Gegenstück zu EXPORT); aus oben genannten Gründen verwendet man auch hier das Paar: Modulname – Funktionsname() bzw. Typ + Variblenname.

❐ WICHTIG: Wichtige Mitteilung: »Programm läuft stabil«, »Fehler in Funktion xy() - Auf nach Indien..«, usw.

❐ DOKU: (Dokumentation) Beschreibung des Moduls. Was macht das Modul, d.h., welche Art von Funktionen enthält es, Besonderheiten?

❐ STATUS: Ein Kreuz in einem oder mehreren Feldern kennzeichnet den Bearbeitungszustand eines Moduls oder einer Funktion:
– <X> Released, Modul/Funktion wurde freigegeben, es gab keine Änderungen
– <X> Version-Update, lediglich die Versions-Nummer wurde auf einen »runden« Wert gebracht (z.B. im Zuge der Freigabe).
– <X> Actual, Modul/Funktion ist aktuell und lauffähig.
– <X> Changing, Modul/Funktion wird gerade geändert und ist evtl. nicht lauffähig.

❐ HIST/BUGS (History – Entstehung und Fehler): Die Entstehungsgeschichte des Moduls in Verbindung mit der Fehlerdokumentation. Entsprechend den Änderungen wird der ZUSTAND und die Versions-Nummer angepaßt.

Zum einen wird hier beschrieben, wann, wer an welchen Funktionen Änderungen vorgenommen hat, und mit einem Stichwort, was geändert wurde – Genaueres findet man dann bei der jeweiligen Funktion.

Zum anderen werden Fehler, die beim Testen des Moduls aufgetreten sind, dokumentiert. Gerade, wenn Tester und Programmierer nicht ein und dieselbe Person sind, können die Fehler an dieser fest definierten Stelle beschrieben werden. Dabei wird die fehlerhafte Version, das aktuelle Datum, das Namenskürzel (s. AUTOR) und eine Fehlerbeschreibung eingetragen. Neue Fehler, wie auch neue Programmversionen werden jeweils ÜBER den alten beschrieben (das Aktuellste immer oben).

❐ AUTOR: Name des Autors und sein Namenskürzel in eckigen Klammern. Weitere Autoren werden hier angefügt.

❐ UMGEBUNG: Welche Sprache, Compiler, Assembler oder Linker wurde verwendet. Evtl. Besonderheiten des Entwicklersystems, welche die Modulfunktionen beeinflussen könnten.

Modulaufbau:

Um im Modul selbst Ordnung zu halten, kann es in die, in der Abbildung vorgeschlagenen Untereinheiten (INCLUDES, TYPEDEFS, DEFINES, usw.) aufgeteilt werden.

Diese Unterteilung können nach eigenen Wünschen als Formular aufgebaut, gespeichert und, wie beschrieben, benutzt werden.

Übersichtlichkeit gewinnt die Aufteilung, wenn die einzelnen Überschriften, wie im Beispiel-Modul »STRINGS.C« geschehen – als dicke Balken (z.B: /*##...#### INCLUDE */) abgesetzt werden. Des weiteren sollten in einem Modul nur diejenigen Teile dokumentiert werden, die auch wirklich vorhanden sind.

Das gilt auch für die Felder der hier vorgestellten Formulare – um keine Mißverständnisse aufkommen zu lassen: Im Formular selbst bleiben natürlich alle Felder erhalten, aber für den einzelnen Modulkopf bzw. für eine spezielle Funktion können Sie sich die Freiheit nehmen, unbenutzte Felder zu löschen.

Die Funktionsdokumentation:

Die Aufgabe einer Funktionsdokumentation ist es, dem Benutzer (auch dem Autor!) einen schnellen und kompletten Überblick über die Funktion zu verschaffen. An einer Funktion interessiert den Benutzer meist weniger, wie sie tatsächlich implementiert wurde. Aus diesem Grund ist für eine effektive Nutzung eine gute Schnittstellenbeschreibung unerläßlich.

Sie fängt beim Namen der Funktion an: Aus praktischen Gründen kürzt man die Funktionsbezeichner oft ab, relativ selten kommen dabei wirklich »sprechende« Abkürzungen zustande. Vorteilhaft ist es, wenn der Benutzer sich unter einer Abkürzung etwas – und zwar das Richtige – vorstellen kann. Ein typisches Beispiel wäre eine Funktion »Tab2Blnk()«: Als Kommentar gehört hier, obwohl diese Abkürzung recht einleuchtend ist (oder nicht?), der Name im Klartext dahinter, also etwa »Convert Tabulators into Blanks«.

Die beim Aufruf der Funktion benötigten Parameter bilden den zweiten Punkt in einem derartigen Interface: Zunächst sollte man für jeden verwendeten Parameter eine neue Zeile spendieren; eine gute Namensgebung hilft dem Benutzer, die Funktion richtig zu verwenden.

Um beim Beispiel zu bleiben: Bei einer Parameterliste »Tab2Blnk (String1, String2)« kann man nicht auf Anhieb sagen, in welchem der beiden Zeichenketten die Tabulatoren ersetzt werden. Das wird eindeutiger, wenn Sie schreiben: »Tab2Blnk (SourceString, NewString)« – verwenden Sie bei den Funktionsparametern Abkürzungen, sollten Sie diese natürlich auch erklären.

Abschließend gehört zu jeder Parameterbeschreibung noch ein Kommentar, in dem folgendes geklärt wird: Was bedeutet die (evtl. verwendete) Abkürzung, welche Funktion hat der Parameter, welche Werte kann er besitzen.

Auf den Rückgabewert, den einige vielleicht schon vermissen, kommen wir gleich zu sprechen; zuvor erst einmal eine kurze Zusammenfassung: Die Beipielfunktion sieht bisher folgendermaßen aus (C-Syntax):

```
int Tab2Blnk     /* convert Tabulators
                    into Blanks */
(char *SrcStr /* Source String - original
                 string, to be converted */
,char *NewStr /* New    String - string
                 with all tab's converted   */
)
```

Vergleicht man dies mit dem zweiten Teil des Funktionsformulars, beginnend mit der Zeile

```
? ??  /* ? */
```

stellt man fest, daß anstelle jedes Fragezeichens ein wohldefinierter Texte zu finden ist.

Der Funktionskopf:

So bedeutet das erste Fragezeichen in der ersten Zeile den Rückgabetyp der Funktion, das Doppelfragezeichen wird durch den Funktionsnamen und das letzte durch den Funktionsnamen im Klartext (bei Abkürzungen) ersetzt. Betrachten wir nun die nächsten Zeilen:

```
 ?   /* ? */
,?   /* ? */
,?   /* ? */
```

Sie sind für die Parameterliste gedacht. Das erste Fragezeichen wird durch den Typ + Name der Variable ersetzt, das Fragezeichen in den Kommentarzeichen steht für die oben erläuterte Kurzbeschreibung. Daß das Komma vor dem Fragezeichen der nächsten Zeile steht, erleichtert im Bedarfsfall das Vervielfältigen der Zeile. Hat eine Funktion z.B. fünf Parameter, wird diese Zeile eben zweimal kopiert.

Prototypen:

Benötigen Sie, wie es in C der Fall ist, Prototypen für die Funktionsdeklaration, müssen Sie lediglich den eben erstellten Routinen-Kopf kopieren, ihn an der gewünschten Stelle einfügen und einen Strichpunkt hinter die schließende Klammer setzen – fertig ist der Prototyp, vergleichen Sie bitte den Prototypen mit der Funktionsdefinition im Modul »STRING.C«.

Man könnte den Prototyp der obigen Funktion auch wie folgt deklarieren:

```
int Tab2Blnk (char*, char*);
```

Nun, entscheiden Sie selbst, welcher von beiden der nützlichere ist.

Übersicht: Was macht die Funktion?

Die Funktionsdokumentation ist gegliedert in Dokumentation, Funktionskopf und Funktionsrumpf. Kopf und Rumpf sollten nicht durch die Dokumentation getrennt werden, denn oftmals muß man beim Codieren einen Blick auf die Definitionen werfen:

❐ RÜCKGABE...: (Rückgabewert)

Der Rückgabewert wird häufig dazu benutzt, dem aufrufenden Programm mitzuteilen, ob die Ausführung erfolgreich war oder nicht. Es sollte zusätzlich die Bedeutung des Rückgabewerts erklärt werden. Für unser obiges Beispiel lautet der Text etwa folgendermaßen:

```
RÜCKGABE: 0 bis (Zahl ersetzter Tab's),
                 falls erfolgreich
             1, sonstige Fälle
```

❐ BEDINGUNG: (Ablaufsbedingung)

Stellt eine Funktion Forderungen an seine Ablaufumgebung, werden diese hier formuliert, d.h., unter welcher Bedingung läuft die Routine fehlerfrei, was setzt sie als gegeben voraus? Im Beispiel »Tab2Blnk()« sieht das so aus:

```
BEDINGUNG.:
   'SrcStr': er darf keinen Null-Pointer
             enthalten; Zeichenkette muß
             mit '\0' beendet werden.
   'NewStr': vor Beenden des Programms muß
             der Speicherplatz, der für die
             konvertierte Zeichenkette
             allokiert wurde, mit 'free()'
             freigegeben werden.
```

❐ FUNKTIONEN:

Häufig rufen Unterprogramme andere Funktionen auf, wobei wir hier drei Arten von Funktionen unterscheiden können: Erstens die globalen, modulfremden Funktionen, zweitens die vom eigenen Modul exportierten und drittens die modulinternen Hilfsfunktionen.

Entsprechend kann hier noch eine weitergehende Unterteilung dieses Punktes gewählt werden.

Das Fragezeichen ist, wie gehabt, durch den Modul-Namen und den Namen der Routine zu ersetzen, z.B.:

```
FUNKTIONEN:  BEISPIEL.C - CountTabs()
             BEISPIEL.C - BlankString()
             MODUL_X.C  - FunktionY()
```

❐ VARIABLEN:

Analog zu den Funktionen kennen wir auch dieselben drei Arten von Variablen: global, modulglobal und modulintern. Auch hier wird das Fragezeichen durch den Namen des Moduls und der Variable ersetzt. Man könnte hier, wie unten gezeigt, auch den Typ der Variablen mitaufnehmen; unserer Meinung nach ist dies jedoch bei Compilern mit Typprüfung nicht notwendig – dafür ist er ja schließlich da.

```
VARIABLEN:   GLOBAL.C   - int  TabWeite
             GLOBAL.C   - char BlankCode
             MODUL_X.C  - char TabCode
```

```
/*
################################################################################

  MODUL......: STRINGS.C
  EXPORT FNKT:
               STRINGS.C - Tab2Blnk()     Convert Tab's into Blanks
  -----------------------------------------------------------------------------
  IMPORT FNKT: BEISPIEL.C - int  CountTabs()
               BEISPIEL.C - void BlankString()
               MODUL_X.C  - int  FunktionY()

  IMPORT VARS: GLOBAL.C   - int  TabWeite
               GLOBAL.C   - char BlankCode
               GLOBAL.C   - char TabCode
  -----------------------------------------------------------------------------
  WICHTIG....: Tab2Blnk() ist nur ein Muster und daher nicht lauffähig
  -----------------------------------------------------------------------------
  DOKU.......: Bereitstellen von Funktionen, die das Manipulieren von Zeichen-
               ketten erlauben.

  STATUS.....: <X> Actual          <X> Released
               < > Changing        <X> Version update

  HIST/BUGS..: Vers/TT-MMM-JJ/Wer   Was
               1.00/01-Dez-92/Ka    Released
               0.02/12-Nov-92/Ka    Tab2Blnk() - dynamisches Allokieren
               0.01/11-Nov-92/Ka    Tab2Blnk() - Fehler behoben
               0.00/10-Nov-92/Ka    neu: Tab2Blnk()
  -----------------------------------------------------------------------------
  AUTOR......: Matthias Straß       [st]
  COPYRIGHT..: (c) 1993

  UMGEBUNG...: Lattice C Compiler V5.10b
               Lattice BLink      V5.10b

################################################################################
*/

/******************************************************************** MAKE */
/*----------------------------------------------------------------- options */
/*
   lc -fit -L STRINGS.C       (so, oder so ähnlich)
*/

/**************************************************************** INCLUDES */
#define MOD_STRINGS
/*------------------------------------------------------------------- amiga */
#include <exec/types.h>
/*------------------------------------------------------------------- std_c */
#include <stdio.h>
#include <string.h>
/*------------------------------------------------------------------ module */
#include "Module_X.h"
#include "Gemeinsam.h"

/***************************************************************** DEFINES */
/*------------------------------------------------------------------ module */
#define MAXLEN 256
/**************************************************************** TYPEDEFS */
/*------------------------------------------------------------------ module */
typedef struct
{  /* beschreibt einen String */
   int len;
   char str[MAXLEN];
}  STRING;

/****************************************************************** EXPORT */
/*--------------------------------------------------------------- functions */
int Tab2Blnk      /* convert Tabulators into Blanks */
(
   char *SrcStr   /* Source String - original string, to be converted */
  ,char *NewStr   /* New    String - string with all tab's converted  */
);
/*--------------------------------------------------------------- variables */
struct IntuitionBase *IntuitionBase;
STRING Error;     /* enthält Fehlermeldungen der Routinen */

/****************************************************************** IMPORT */
/*--------------------------------------------------------------- functions */
int   CountTabs
(  char *s);              /* string with 0 or more tabs */

void BlankString
(  STRING *s);            /* string to be blanked */

int FunktionY (void);     /* it's doing something... */
/*--------------------------------------------------------------- variables */
extern int  TabWidth;       /* in char's */
extern char BlankCode;      /* char, used by BlankString to emty a string */
extern char TabCode;        /* char, to make a tab visible */

/*
################################################################################
################################################################ IMPLEMENTATION
################################################################################
*/

/*
############################################################ GRUPPE: STRING-UMWANDLUNGEN
*/

/************************************************************** Tab2Blnk()

  RÜCKGABE...:  0 bis (Anzahl der ersetzten Tab's), falls erfolgreich
               -1, sonst
  BEDINGUNG..: 'SrcStr': er darf keinen Null-Pointer enthalten; die
                         Zeichenkette muß mit einem '\0' beendet werden.
               'NewStr': vor Beenden des Programms muß der Speicherplatz,
                         der für die konvertierte Zeichenkette allokiert
                         wurde, mit der Systemfunktion 'free()' freigegeben
                         werden.
  -----------------------------------------------------------------------------
  FUNKTIONEN.: BEISPIEL.C - int  CountTabs()
               BEISPIEL.C - void BlankString()
               MODUL_X.C  - int  FunktionY()
  VARIABLEN..: GLOBAL.C   - int  TabWeite
               GLOBAL.C   - char BlankCode
               GLOBAL.C   - char TabCode
  -----------------------------------------------------------------------------
  DOKU.......: Bereitstellen von ausreichend Speicherplatz für 'NewStr'.
               Es wird 'SrcStr' nach 'NewStr' kopiert und dabei nach
                  Tabulatorzeichen ('TabCode') gesucht. Sie werden in 'NewStr'
                  durch die in 'TabWeite' angegebene Anzahl Leerzeichen
                  ('BlankCode') ersetzt.
               'NewStr' zeigt auf die neu entstandenen String.

  LÖSUNG.....: (1) Füllen meines Hilfsstrings mit Leerzeichen
               (2) Mit dem ersten Zeichen aus 'SrcStr' beginnen
               (3) Ist das Zeichen ein 'TabCode', geschieht dies-und-das
               (4) usw.
  -----------------------------------------------------------------------------
  HIST/BUGS..: Vers/TT-MMM-JJ/Wer   Was
               1.00/01-Dez-92/St    Released
               0.20/12-Nov-92/St    Speicherplatz wird jetzt dynamisch
                                    allokiert
               0.10/11-Nov-92/St    Fehler beim Hochzählen behoben
               0.00/10-Nov-92/St    Fehler beim Hochzählen ...
               0.00/10-Nov-92/St    Ersterstellung
-------------------------------------------------------------------------------
*/
int Tab2Blnk      /* convert Tabulators into Blanks */
(  char *SrcStr   /* Source String - original string, to be converted */
  ,char *NewStr   /* New    String - string with all tab's converted  */
)
{
  char *hilfString;
  short tabCounter;
  short index;
  ...
  /* hier folgt der Code mit den Nummer aus dem Punkt "LÖSUNG" an den */
  /* entsprechenden Stellen */
  ...
  for (index=0; index < MAX; index++)
  {   /* sollte hier stehen, was der Block macht (logische Einheit) */
      /* (2)   (siehe oben)*/
      ...

      if (hilfsString[index]==TabCode)
      {  /* (3), Tab durch Blanks ersetzen */
         ...
      }

      ...

   }  /* for (so weiß ich auch bei langen Blöcken, das dies eben der */
      /*      FOR-Block ist*/
   ...
} /************************************************************ Tab2Blnk() */

/*####################################################### end of module STRINGS */
```

Formular: Ein Beispiellisting, das zeigt, wie Sie Ihre Programme dokumentieren sollten.

❐ DOKU...: ? (Dokumentation)
Es erfolgt eine Beschreibung des Unterprogramms. Was macht die Funktion? Was nicht? Beispiel:

```
DOKU...: Bereitstellen von ausreichend
         Speicherplatz für 'NewStr'.
         Es wird 'SrcStr' nach 'NewStr'
         kopiert und dabei nach Tabulatorz
         eichen ('TabCode') gesucht. Sie
         werden in 'NewStr' durch die in
         'TabWeite' angegebene Zahl Leer-
         zeichen ('BlankCode') ersetzt.
         'NewStr' zeigt auf neuen String.
```

❐ LÖSUNG...:
Ist der verwendete Algorithmus besonders trickreich, ausgefeilt oder kompliziert, sollte er hier beschrieben werden. Das erleichtert dem Programmierer später viel Kopfzerbrechen, wenn es darum geht, den Ablauf der Funktion nachzuvollziehen.

Gerade dann, wenn die Lösung komplex ist, erweist es sich als vorteilhaft, den Algorithmus zu beschreiben und dabei die einzelnen Schritte durchzunumerieren. Im Programm-Code selbst verwendet man dann anstatt des Textes lediglich diese Nummern, um den Code nicht unnötig aufzublähen:

```
LÖSUNG...:
   (1) Füllen meines Hilfsstrings mit
       Leerzeichen
   (2) Beginnend mit dem ersten Zeichen
       aus 'SrcStr' nach Tab's suchen
   (3) Ist das Zeichen ein »TabCode«,
       geschieht dies-und-das
   (4) usw.
```

Erklären Sie Ihre Lösung

❐ ZUSTAND:
Hier möchten wir auf denselben Punkt aus dem Modulformular verweisen. Für die Tab2Blnk()-Funktion sieht das so aus:

```
ZUSTAND....: <X> Released
<X> Version update  <X> Act  < > Changing
```

❐ HIST/BUGS: Dies ist derselbe Punkt wie beim Modulformular (siehe Abbildung). Wichtig ist bei diesem Punkt, daß der aktuellste Eintrag immer oben steht! Unter »Was« wird notiert, »was« erweitert, geändert oder verbessert wurde.

```
ENTSTEHUNG:
 Vers/TT-MMM-JJ/Wer      Was
 1.00/12-Dez-91/Ka    Released
 0.10/12-Dez-91/Ka    Speicherplatz wird
                      dynamisch allokiert
 0.01/11-Dez-91/Ka    Fehler beim Hoch-
                      zählen des Index beh.
 0.00/10-Dez-91/ka    Ersterstellung
```

Eine letzte Bemerkung betrifft die System- bzw. Library-Funktionen. Verwendete Routinen dieser Art werden aus zweierlei Gründen nicht aufgeführt: Zum einen gehören diese Funktionen meist fest zum System – wie der Name schon sagt, zum anderen sind sie ausgetestet und lauffähig (hoffentlich), so daß Änderungen an diesen Funktionen höchstens die Performance, nicht aber die generelle Funktion, geschweige denn die Parameter betreffen.

Namensgebung bei Variablen und Funktionen:
Ausgehend von Compilern, die während der Übersetzungsphase Typprüfungen durchführen, bietet sich folgende »Strategie« an: Lokale Variablen/Funktionen beginnen generell mit einem kleinen Anfangsbuchstaben, globale Variablen/Funktionen mit einem Großbuchstaben. Die Namen werden evtl. aus mehreren Worten zusammengesetzt, wobei jedes weitere Wort mit einem Großbuchstaben beginnt. Beispiele:

```
lokal:
 short idxArrayA    - Index eines Feldes A
 long  counter
 short countTabs()  - Hilfsfunkt. zum Zählen
                      von Tab's
global:
 short CountTabsInString() - wie oben,
                             aber global
 char  TabCode - ASCII-Code des Tab's
 long  Diff - eine Differenz
```

Es ist nicht sinnvoll – weil Ballast – bei den Variablen/Funktionen den Typen in irgendeiner Form mitzuführen (als Präfix o.ä). Schließlich ist es die Aufgabe des Compilers, die Typzuweisungen zu überprüfen.

In Systemen, in denen keine Typprüfung stattfindet, ist es, zur Selbstkontrolle, hilfreich, die Typen beispielsweise als Präfix mitzuführen. So könnte man jeden Typen mit einem Buchstaben abkürzen und mit einem Unterstrich vor die "Variablennamen stellen (»i_« für integer, »s_« für String, usw.). Wichtig ist einzig und allein, daß irgendwo, an einer zentralen Stelle (am besten im Modulkopf) beschrieben wird, welche Abkürzung was bedeutet, und daß es konsequent eingehalten wird. Wie die einzelnen Abkürzungen tatsächlich lauten, ist im Prinzip unwesentlich.

Kommentierung:
Ganz allgemein muß bei Kommentierungen im Quelltext darauf geachtet werden, daß man nicht jeden einzelnen Befehl beschreibt; eine schlechte Kommentierung ist z.B. folgende:

```
i=START;          /* 'i' auf START setzen */
          /* solange 'i' kleiner als ENDE */
while (i <= ENDE)
{      /* setze das 'i'-te Element auf 0 */
   Feld[i] = 0;
   /* erhöhe 'i' um 5 */
   i += 5;
}
```

Untersuchungen (B. Sheiderman, »Software Psychology«, 1980) zeigen, daß derartige »Kommentare« nicht zum Verständnis des Algorithmus beitragen. Statt dessen sollten Zusammenhänge, logische Einheiten erläutert werden. Beispiel:

```
i=START;
while (i <= ENDE)
{          /* Löschen jedes 5. Elements */
   Feld[i] = 0;
   i += 5;
}
```

Derartige Kommentare sind häufig auch kürzer und übersichtlicher als die »Low-Level-Kommentierung« (vergleiche hierzu das obige Code-Fragment).

Zusammenfassung:
In der Beispiel-Dokumentation finden Sie die meisten der ausgeführten Punkte noch einmal im Zusammenhang (siehe Listing). Dort wird zusätzlich noch ein Vorschlag bzgl. der Einrück-Strategie gemacht: Öffnende und schließende Klammern sollten übereinander stehen, der Block zwischen den Klammern um einen – nicht zu großen – Tabulatorschritt eingerückt werden. Jede öffnende Klammer, die ja einen Block einleitet, sollte sofort mit einem Kommentar versehen werden.

Dokumentation: Die Mühe lohnt sich

Motivationen:
Das alles mag jetzt, für den einen oder anderen Leser etwas zuviel des Guten sein. Aber wie bei allen Dingen, die das Programmieren betreffen, gilt vor allem hier: Ausprobieren.

Tippen Sie die beiden Formulare ab und verwenden Sie sie bei nächster Gelegenheit. Wissen Sie nicht mehr genau, was in dem einen oder anderen Feld beschrieben werden soll, schauen Sie im Artikel nach. Sie werden sehen, daß Sie schon sehr bald auf die Beschreibungen ganz verzichten können.

Vergleichen Sie auch einmal Ihre bisherigen Dokumentationen mit dem hiesigen Vorschlag, am besten, wenn Sie schon eine Funktion oder vielleicht sogar ein Modul auf die neue Weise dokumentiert haben.

Haben Sie Fragen, Anregungen, Kommentare? Der Autor freut sich über jede Reaktion. Sie finden seine Adresse im Anschluß an diesen Artikel. Sie können Ihre Anmerkungen aber natürlich auch unsere Redaktion schicken ■

Matthias Straß, Wöhrder Hauptstr. 30, 90489 Nürnberg
Besonders danken möchten wir Bernd »Nicky« Koll, der die Formulare mit dem Autor überarbeitet hat.

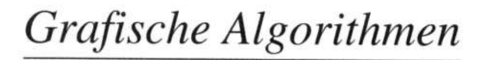
Der Autor über sich

Matthias Straß:

Geboren: 07. 01. 69. Seit ich 15 Jahre alt bin, komme ich von den Computern nicht mehr los.

Alles fing mit einem harmlosen C-64er an. Nach dem Gymnasium begann ich 1989 an der FH Nürnberg das Informatik-Studium. Zu diesem Zeitpunkt machte ich die ersten Erfahrungen mit dem Amiga 500. Inzwischen bin ich bei einem Amiga 2000 gelandet.

Ich programmiere hauptsächlich (meist in C, hin und wieder in Assembler) oder erstelle mit TeX Referate für diverse Vorlesungen.



e=mc

Mathematische Th

Haben Sie sich schon einmal überlegt, wie man auf dem Computer eine Linie zeichnet? Viele werden jetzt sagen: »Ganz einfach. Dafür gibt es doch den Befehl 'Line'«. Doch welche Algorithmen werden dabei verwendet? Dieser Artikel erklärt auf einfache Art und Weise die erforderlichen mathematischen Grundlagen und vergleicht einige Routinen, um Linien zu ziehen.

von Bernd Wiedemann

Das Bild des Amiga setzt sich aus Punkten (Pixeln) zusammen. Jeder kann einzeln angesprochen werden, z.B. durch die Funktion »WritePixel«, die man in der »graphics.library« findet.

Will man nun zwei Punkte durch eine Linie verbinden, treten sofort die ersten Probleme auf: Es ist nämlich bis auf drei Ausnahmen unmöglich, eine gerade Linie auf dem Computer darzustellen.

Wie man in Bild 1 unten sieht, kann man die Linie nur durch geschickte Auswahl von Punkten annähern (approximieren). Doch diese Aufgabe soll uns der Computer abnehmen!

Gerade Geraden – gar nicht leicht

Was wir brauchen, ist ein entsprechender Algorithmus. Doch bevor wir diesen erläutern, sollte man sich darüber im klaren sein, welche Eigenschaften er überhaupt haben sollte:

1. Gerade Linien sollten auch als gerade Linien erscheinen.
2. Die Helligkeit sollte konstant sein, und zwar unabhängig von der Länge und Steigung der Geraden.
3. Die Linie sollte so schnell wie möglich gezeichnet werden.
4. Linien sollten genau abgebildet werden, d.h. sie sollten genau starten und enden.

Leider kann kein Algorithmus diese vier Eigenschaften aufweisen, denn da der Amiga, wie schon erwähnt, mit Rastergrafik arbeitet, kann keine Linie gerade sein. Eine Ausnahme bilden vertikale, horizontale und 45-Grad-Linien (Bild 2).

Die Helligkeit der Linie ist ebenfalls nur bei den drei Spezialfällen konstant. Allerdings ist sie auch hier abhängig von der Steigung. Die vertikalen und horizontalen Linien leuchten nämlich stärker als die 45-Grad-Linie. Dies ist deshalb so, weil der Zwischenraum zweier benachbarter Pixel auf der 45-Grad-Linie größer ist als auf den beiden anderen Linien.

Zu Punkt 3 bleibt nur zu sagen: Das Zeichnen von Linien kann, wie sollte es auch anders sein, nie schnell genug gehen. Allerdings kann man einen Algorithmus enorm beschleunigen, indem man nur Ganzzahl-Arithmetik (Integer) verwendet, d.h. man verzichtet auf Divisionen und Fließkommazahlen.

Will man allerdings die Linien genau abbilden, ist ein gewisser Mehraufwand erforderlich. Und das bedeutet wiederum eine Zunahme der Rechenzeit.

Doch nun gehts »ans Eingemachte«. Sehen Sie sich dazu bitte Listing 2 auf der übernächsten Seite an. In diesem und in allen darauffolgenden Programmen wird auf das Include-File »Grafik-Grundlage.h« (siehe Listing 1 nächste Seite) zurückgegriffen. Es stellt die für uns erforderlichen Bedingungen her.

Zuerst werden die benötigten Libraries geöffnet. Danach initialisiert die Funktion »Vorbereitung« einen Bildschirm (Screen) und ein Fenster (Window). Die Funktion »Ende« dagegen schließt diese und auch die verwendeten Libraries. Dadurch, daß das File in jedes folgende Programm eingebunden wird, sparen Sie viel Schreibarbeit.

Alle Programme sind in C geschrieben und wurden auf dem Aztek-C-Compiler V.3.6 getestet. Allerdings sollte es auch mit dem SAS- bzw. Lattice-Compiler keine Probleme geben.

Nachdem Sie die Listings hoffentlich fehlerfrei eingegeben haben, werden sie mit dem Befehl »cc NAME +ff« kompiliert, wobei man für »NAME« den richtigen Namen einsetzen sollte. Danach gibt man noch das Kommando »ln NAME -lm -lc« ein und schon hat man ein lauffähiges Programm, das man mit einem Mausklick auf das Close-Gadget auch bequem verlassen kann.

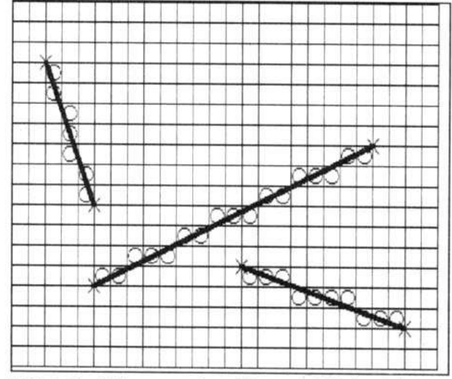


Bild 1: Wegen des Rasterbildschirms kann auf dem Amiga keine gerade Linie dargestellt werden. Da man sie nur annähern kann, entstehen die lästigen Treppeneffekte.

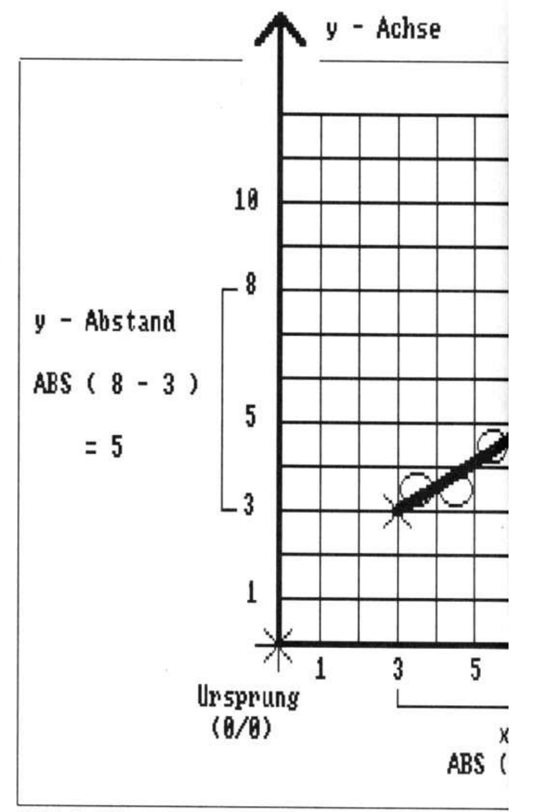


Bild 3: Die Linie besteht aus neun Pixeln, di

Digital Differential Analyzer (DDA)

Doch zurück zu Listing 2. Es enthält einen ganz einfachen Linienalgorithmus, den sog. Digital Differential Analyzer, kurz DDA. Dieser basiert auf der rekursiven Programmierung und auf der Methode der Inkrementierung (Steigerung).

Um eine Linie von P(x1;y1) nach Q(x2;y2) überhaupt zeichnen zu können, muß man wissen, aus wievielen Punkten sie besteht, d.h. wieviel Punkte gesetzt werden müssen, um sie optimal anzunähern. Da die beiden Endpunkte P und Q bekannt sind, wird durch folgende Gleichung die Länge approximiert.

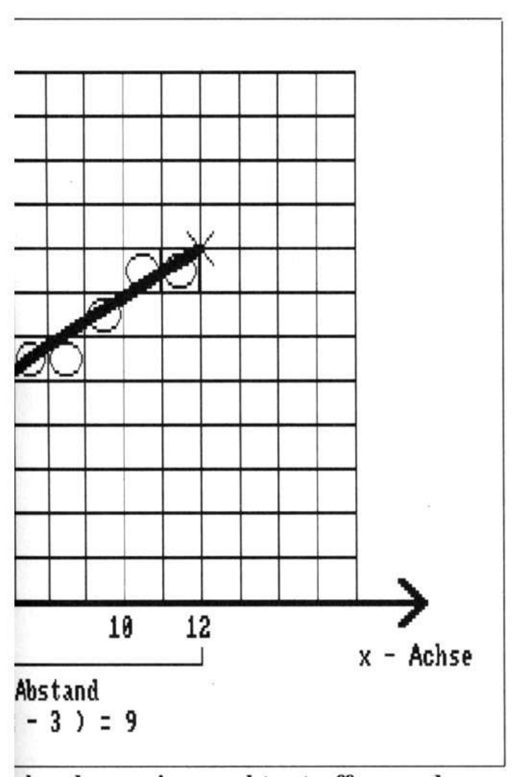
```
LENGTH = MAX(ABS(x2-x1),ABS(y2-y1))
```

d.h. der größere der beiden Abstandswerte wird zur weiteren Berechnung herangezogen.

Zum Kern des Algorithmus: Er geht von seinem Anfangspunkt P(x1;y1) aus und schließt von diesem auf den benachbarten Punkt P1, der die Linie am besten approximiert. Rekursive

orien in der Praxis

$a^2+b^2=c^2$



ehr oder weniger exakt getroffen werden.

Programmierung nennt man das. Von P1 hangelt man sich folglich nach P2, von P2 nach P3 ... Dies geht solange, bis man am Punkt Q(x2;y2) angelangt ist.

Anhand eines Beispiels ist dies am leichtesten verständlich. Angenommen, es soll eine Linie vom Punkt P(3;3) zu Punkt Q(12;8) gezogen werden. Wie man in Bild 3 (links) leicht erkennt, beträgt die Länge der Linie neun Pixel (x2-x1 = 12-3 = 9).

Der erste Pixel hat die Koordinaten x=3 und y=3. Doch welche hat der benachbarte Pixel? Da die Variable Length = ABS(x2-x1), d.h. der X-Abstand größer als der Y-Abstand ist, und x2 größer als x1 ist, wird die Linie in Richtung der positiven X-Achse gezeichnet, d.h. von links nach rechts. Das bedeutet, daß der nächste Punkt immer eine um 1 größere X-Koordinate hat als der vorhergehende. Die Schrittweite dx in X-Richtung ist also 1.

Und welche Y-Koordinate hat der Pixel? Hierzu benötigt man wieder die Schrittweite, jetzt jedoch dy. Sie sagt aus, um wieviel die Y-Koordinate des Pixels von der Y-Koordinate des vorhergehenden abweicht. dy läßt sich zum Glück leicht berechnen.

```
dy = (y2-y1)/LENGTH
```

Im Beispiel enthält dy den Wert 5/9. Der zum Punkt P(3;3) benachbarte Pixel müßte also die Koordinaten P1(4;3+5/9) haben. Da Pixel allerdings nur ganzzahlige Koordinaten haben können, wird der Nachkommateil eliminiert. Dies geschieht mit Hilfe der SPFloor-Funktion. Sie liefert die größte ganze Zahl, die kleiner als die angegebene ist, in diesem Falle die Zahl 3. Somit hat P1 die Koordinaten (4;3). Der nächste Punkt P2 hätte die Koordinaten (5;4+1/9), also genau (5;4).

Nimmt man nun eine andere Linie, kann es sein, daß sie in Richtung der negativen Y-Achse gezeichnet wird, d.h. LENGTH = ABS(y2-y1), da ABS(y2-y1) > ABS(x2-x1), und y2 ist kleiner als y1, so ist dy = -1. Jetzt muß man also dx berechnen. Man verwendet dabei fast genau die gleiche Formel wie für dy.

```
dx = (x2-x1)/LENGTH
```

In diesem Beispiel werden die Y-Koordinaten fortlaufend um -1 und die X-Koordinaten um dx erhöht.

Damit der Algorithmus auch in allen Quadranten des Koordinatensystems korrekt arbeitet, muß man die Variablen x und y folgendermaßen initialisieren:

```
x = x1 + 0.5*SPTst(dx)
```

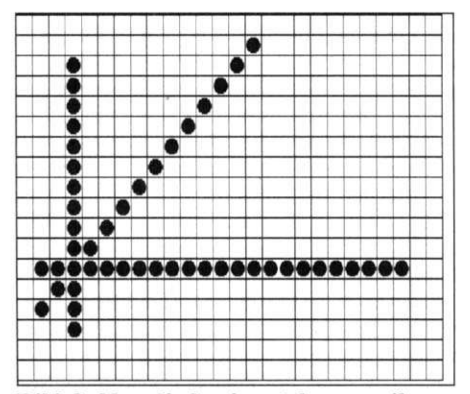


Bild 2: Nur die horizontalen, vertikalen und 45-Grad-Linien können mit Recht als gerade bezeichnet werden.

```
#include <intuition/intuitionbase.h>
#include <libraries/mathffp.h>
/* Nächste Zeile nur eingeben, falls ein */
/* Aztek-C-Compiler verwendet wird.      */
#include <functions.h>
struct IntuitionBase *IntuitionBase;
struct GfxBase       *GfxBase;
struct Screen        *screen;
struct Window        *window;
/* Daten für den Screen: Auflösung 320 * 256 Pixel  */
/*                      und 1 Bitplane              */
struct NewScreen  screendaten =
      {0,0,320,256,1,0,1,NULL,CUSTOMSCREEN,NULL,NULL,NULL,NULL};
/* Daten für das Window: Auflösung 320 * 256 Pixel  */
/* Das Window kann durch Anklicken des CloseWindow- */
/* Gadgets geschlossen werden.                      */
struct NewWindow  windowdaten = {0,0,320,256,0,1,CLOSEWINDOW,
    ACTIVATE|WINDOWCLOSE,NULL,NULL,(UBYTE *)"GRAFIKWINDOW",
    NULL,NULL,NULL,NULL,NULL,NULL,CUSTOMSCREEN,};
Vorbereitung()
/* Diese Funktion öffnet alle benötigten Libraries, */
/* den Screen und das Window.                       */
   { if(!IntuitionBase)
        { IntuitionBase  =  (struct IntuitionBase *)
                             OpenLibrary("intuition.library",0L);
          if(!IntuitionBase)
             { Ende();
             }
        }
     if(!GfxBase)
        { GfxBase        =  (struct GfxBase *)
                             OpenLibrary("graphics.library",0L);
          if(!GfxBase)
             { Ende();
             }
        }
   screen = (struct Screen*)OpenScreen(&screendaten);
   if(!screen)
      { Ende();
      }
   windowdaten.Screen =  screen;
   window = (struct Window*)OpenWindow(&windowdaten);
   if(!window)
      { Ende();
      }
   return(NULL);
   }
Ende()
/* Diese Funktion schließt alles, was Vorbereitung geöffnet hat */
   { if(window) CloseWindow(window);
      if(screen) CloseScreen(screen);
      if(GfxBase) CloseLibrary(GfxBase);
      if(IntuitionBase) CloseLibrary(IntuitionBase);
      return(NULL);
   }
```


Listing 1: »Grafikgrundlage.h« – das Include-File dient als Ausgangsbasis unserer Testprogramme

```
y = y1 + 0.5*SPTst(dy)
```

Die Funktion SPTst(x) liefert die Zahl 1, falls x positiv ist, die Zahl -1, falls x negativ ist, und die Zahl 0, falls x=0. Sie ist identisch mit der Signum-Funktion aus der Mathematik.

Allgemein kann man allerdings sagen, daß der Algorithmus sehr genau arbeitet. Diesen Vorteil erkauft man sich aber mit einem gravierenden Nachteil. Da der DDA-Algorithmus mit Fließkommazahlen arbeitet und zudem auch noch Divisionen durchführt, kann man ihn wahrlich nicht als schnell bezeichnen. Im Gegenteil, startet man das Programm »DDA« und paßt gut auf, dann kann man sogar noch gut erkennen, wie die Linie gezeichnet wird.

Noch eine Anmerkung: Benutzt man anstatt der verwendeten »SPFloor«-Funktion eine andere, beispielsweise die »ROUND«-Funktion, erhält man wiederum andere Ergebnisse.

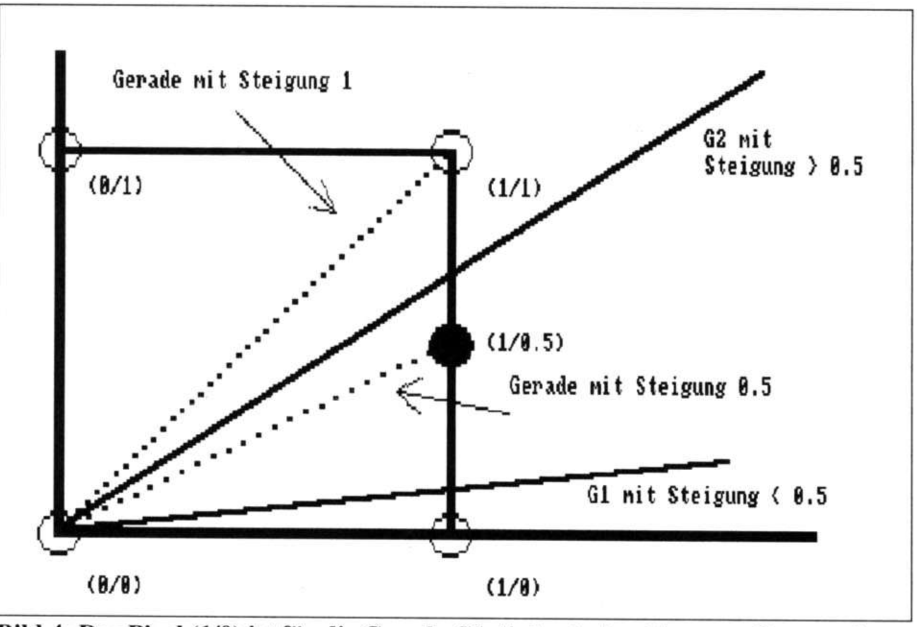


Bild 4: Das Pixel (1/0) ist für die Gerade G1 die beste Annäherung. Ganz anders sieht es jedoch bei G2 aus. Hier wird der Punkt (1/1) herangezogen.

Geschwindigkeit ist Trumpf

Kommen wir zu einem anderen Linienalgorithmus, dem der DDA in puncto Geschwindigkeit nicht das Wasser reichen kann. Es handelt sich um den »Bresenham-Algorithmus«.

Wie beim DDA wird auch hier die Linie entweder in der x- oder in der y-Richtung gezeichnet. Das hängt natürlich von ihrer Steigung ab. Das heißt: Bei jedem Durchlauf wird die x- oder y-Koordinate um 1 verändert. Für die jeweils andere Variable wird der Abstand, auch Fehler oder Error genannt, von der Linie zum nächsten Gitterpunkt berechnet. Dadurch wird entschieden, ob sich auch diese um 1 verändert oder gleich bleibt.

Die Schnelligkeit des Bresenham-Algorithmus beruht dabei u.a. auf folgendem Trick: Es wird nämlich nur das Vorzeichen des Errors ausgewertet. Zum besseren Verständnis sollten Sie sich Bild 4 anschauen. Hier wird allerdings angenommen, daß die Gerade eine Steigung zwischen 0 und 1 hat und durch den Ursprung (Koordinaten 0 / 0) geht. Somit ist der erste zu plottende Punkt auch schon gefunden. Da die Linie aufgrund der Steigung in Richtung der positiven x-Achse gezeichnet wird, hat das nächste Pixel die x-Koordinate 1. Doch welche y-Koordinate soll es haben? Diese Frage läßt sich leicht beantworten. Ist die Steigung der Geraden kleiner als 0.5 (wie bei G1), so bleibt sie 0, da der Abstand von der Linie zum Punkt (1/0) kleiner ist als der zum Punkt (1/1) (wie bei G2). Falls die Steigung größer oder gleich 0.5 ist, wird sie 1.

Damit Sie den Bresenham-Algorithmus auch vollständig durchschauen, geben wir Ihnen ein weiteres Beispiel, bei dem auch die Vorzeichenauswertung des Fehlers erklärt wird. Sehen Sie sich hierzu Bild 5 an.

Gegeben ist eine Gerade mit der Steigung 2/7, die durch den Ursprung (0/0) geht. Ebenso wie oben liegt es auf der Hand, daß der erste zu setzende Punkt (0/0) ist. Da die Linie in positiver x-Richtung gezeichnet wird, erhöht man die x-Koordinate bei jedem Durchlauf um 1. Doch nun soll die Vorzeichenauswertung zum Tragen kommen.

Der Trick mit dem Vorzeichen

Um ein korrektes Ergebnis zu erhalten, ist es notwendig den Fehler e mit -0.5 zu initialisieren und bei jedem Durchlauf die Steigung m der Geraden hinzuzuaddieren, d.h.:

```
e = e + m
```

```
/* Digital Differential Analyzer (DDA) */

/* Include-File einbinden, das notwendige Vorbereitungen trifft (List.1)*/
#include <Grafik-Grundlage.h>

   struct RastPort* rasp;

main()
{  int i;                          /* Alle verwendeten   */
   float length;                   /* Variablen sind     */
   float x1,x2,y1,y2,x,y,dx,dy; /* hier aufgeführt.   */

                                   /* Ein Screen, ein Window und     */
                                   /* notwendigen Libraries werden   */
   Vorbereitung();                 /* geöffnet Diese Funktion        */
                                   /* befindet sich im Include-File */
                                   /* "Grafik-Grundlage.h".          */
   rasp=window->RPort;
   x1 = 30;
   y1 = 30;                /* Koordinaten von P(30/30)   */
   x2 = 100;
   y2 = 80;                /* Koordinaten von Q(100/80) */

/* Die angenäherte Länge wird bestimmt */
   if(abs(x2-x1)>=abs(y2-y1))
     { length=abs(x2-x1);
     }
   else
     { length=abs(y2-y1);
     }
   dx = (x2-x1)/length;     /* Schrittweite in X-Richtung */
   dy = (y2-y1)/length;     /* Schrittweite in Y-Richtung */

  /* Die Anfangswerte von x und y garantieren dafür, daß       */
  /* der Algorithmus in allen 4 Quadranten korrekt arbeitet   */
  x  = x1 + 0.5*SPTst(dx); /* SPTst(x) liefert -1, falls x negativ */
  y  = y1 + 0.5*SPTst(dy); /* 0, falls x == 0, +1, falls x positiv */
                           /* Vergleichbar mit SIGNUM (sgn)        */
  /* Die Schleife wird LENGTH mal durchlaufen */
  for( i = 1; i <= length; i++ )      /* Laufvariable i enthält die */
                                      /* Anzahl der Durchläufe      */
  { /* Koordinaten für die Funktion WritePixel müssen ganzzahlige  */
    /* long-Variablen sein. Die Funktion SpFloor(x) liefert die    */
    /* größte ganze Zahl, die kleiner als x ist, wobei x eine      */
    /* FLOAT-Variable ist wie die von der Funktion SPFloor(x)      */
    /* gelieferte Zahl.                                             */
      WritePixel(rasp,(long)SPFloor(x),(long)SPFloor(y));
      x=x+dx;          /* Inkrementierung der */
      y=y+dy;          /* Variablen x und y   */
                       /* Laufvariable i enthält die */
                       /* Anzahl der Durchläufe      */
   }
                                          /* Funktion wartet, bis */
   Wait(1L<<window->UserPort->mp_SigBit); /*   Close-Gadget       */
                                          /*   angeklickt wird.   */
                 /* Das Window, der Screen und alle benötigten */
                 /* Libraries werden geschlossen. Diese        */
   Ende();       /* Funktion befindet sich im Include-File    */
                 /* "Grafik-Grundlage.h"                       */
}
}
```


Listing 2: Wir zeichnen eine Linie mit einem recht einfachen Algorithmus, dem Digital Differential Analyzer

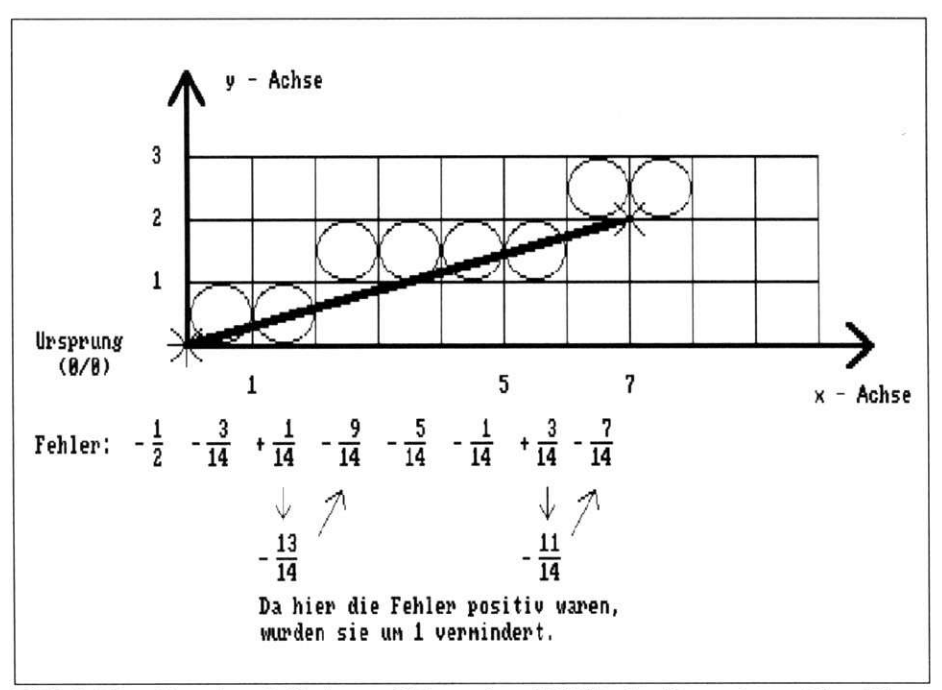


Bild 5: Eine Gerade mit Steigung 2/7 wurde mit Hilfe des Bresenham-Algorithmus gezeichnet. Der Fehler e wurde mit -0.5 initialisiert. Bei jedem Durchlauf wurde die Steigung hinzuaddiert und anschließend das Vorzeichen ausgewertet.

In unserem Beispiel hat e nach dem ersten Durchlauf den Wert

`e = -0.5 + 2/7 = -3/14`

e ist also negativ. Daraus folgt, daß die y-Koordinate des vorhergehenden Punktes unverändert übernommen wird. Somit erscheint auf dem Bildschirm der Punkt (1/0).

Jetzt kommen wir zum nächsten Durchlauf. Wir erinnern uns daran, daß die x-Koordinate um 1 erhöht werden muß, d.h. x = 2 . Ebenso wird zu e wieder die Steigung addiert.

`e = e + m = -3/14 + 2/7 = 1/14`

Da e nun positiv ist, wird die y-Koordinate um 1 erhöht, d.h. y = 1. Wir erhalten den Punkt (2/1). Allerdings muß jetzt der Fehler neu initialisiert werden, denn sonst würde er bei jedem Schleifendurchlauf positiv bleiben, da ja auch die Steigung positiv ist. Das hätte zur Folge, daß ständig die y-Koordinate um 1 erhöht würde. Dies wäre allerdings ein völlig unerwünschtes Ergebnis. Daher muß e um 1 vermindert werden,

`e = e - 1 = 1/14 - 1 = -13/14.`

Doch nun geht es »normal« mit dem Bresenham-Algorithmus weiter. Der nächste Punkt hat die x-Koordinate 3. Der Fehler e wird zu

`e = -13/14 + 2/7 = -9/14.`

Man sieht sofort: e ist negativ, d.h. die y-Koordinate bleibt gleich und auf dem Bildschirm leuchtet das Pixel (3/1) auf. Nach einem weiteren Durchlauf gesellt sich zu diesem der Punkt (4/1), da der Error e = -9/14 + 2/7 = -5/7 wieder negativ ist. Bis zum Ende der Linie geht es nun immer so weiter.

Wir hoffen, daß Sie jetzt mit dem Bresenham-Algorithmus vertraut sind und das dazugehörende Listing 3 langsam auf sich einwirken lassen können. Wenn man es nämlich genauer betrachtet, fällt einem sofort auf, daß dieser Algorithmus, ebenso wie der weiter oben besprochene DDA, mit Divisionen und Fließkommazahlen arbeitet. Mit dem erhofftem Geschwindigkeitsvorteil ist es also bis jetzt noch nichts geworden.

Tempo durch Integer-Arithmetik

Doch dieser Mister Bresenham war ein sehr schlauer Kopf. Aber sehen Sie selbst. Im Listing erscheint die Zeile

`e = dy/dx - 0.5`

Hier muß der Computer lange rechnen. Falls es uns nun gelingen würde, diese Zeile so zu modifizieren, daß nur noch Integer-Arithmetik verwendet werden würde, hätte man die oben so hoch gepriesene Geschwindigkeitssteigerung endlich erreicht.

Und das ist nicht so schwer. Wie wir wissen, kommt es bei dem Fehler e nur auf das Vorzeichen an. Falls wir nun obige Gleichung mit 2 * dx multiplizieren, erhalten wir

`2 * dx * e = 2 * dy - dx`

Da sich das Vorzeichen nicht ändert, führen wir einen »neuen« Fehler e´ ein:

`e´ = 2 * dx * e = 2 * dy - dx`

Wenn man nun den restlichen Programmcode an diese Zeile anpaßt, erhält man aufgrund der Integer-Arithmetik einen sehr schnellen Linienalgorithmus, wie man in Listing 4 (nächste Seite) auch sehen kann. Allerdings funktioniert dieser nur im 1.Oktanten perfekt.

Natürlich wollen wir Ihnen auch den »Allgemeingültigen Bresenham-Algorithmus« nicht vorenthalten. Je nachdem in welchem Oktanten man liegt, wird die x- oder y-Komponente zur Laufvariablen und hier entscheidet sich auch, ob auf- oder abgezählt wird. Doch darauf will ich jetzt nicht näher eingehen. Studieren Sie zum Schluß Listing 5.

Sie sehen, daß sich mit einem »kleinem bißchen« Mathematik doch recht interessante Probleme bewältigen lassen. Aber noch mehr würde es uns freuen, wenn Ihnen dieser Artikel gefallen hat. ■

Literaturhinweis:
David F. Rogers "Procedural Elements for Computer Graphics" McGraw-Hill Book Company ISBN 0-07-053534-5 Seite 29 - 42

```
/* Bresenham-Algorithmus für den ersten Oktanten */

/* Include-File einbinden, das notwendige Vorbereitungen trifft (List.1)*/

#include <Grafik-Grundlage.h>

   struct RastPort* rasp;

main()
{ int i;                          /* Alle verwendeten   */
   float e;                       /* Variablen sind     */
   long x1,x2,y1,y2,x,y,dx,dy;    /* hier aufgeführt.   */

   Vorbereitung();                /* Eine bekannte Funktion   */
                                  /* siehe Grafik-Grundlage.h */
   rasp=window->RPort;
   x1 = 30;
   y1 = 30;               /* Koordinaten von P1 */
   x2 = 100;
   y2 = 80;               /* Koordinaten von P2 */

/* Beginn des Algorithmus */
   x = x1;
   y = y1;
   dx = x2 - x1;
   dy = y2 - y1;

/* Der Fehler e wird initialisiert */
   e =  ((float)dy/(float)dx) - 0.5;

   for(i=1;i<=dx;i++)
      {
      WritePixel(rasp,x,y);
      while(e>=0)         /* Bei positivem e wird y erhöht    */
         {                /* anschließend wird e reinitialisiert */
         y = y + 1;       /* bei negativem e dagegen, wird     */
         e = e - 1;       /* y nicht verändert */
         }
      x = x + 1;
      e = e + ((float)dy/(float)dx);
      }

   Wait(1L<<window->UserPort->mp_SigBit);
   Ende();
}
}  © 1993 M&T
```

Listing 3: Beispiel für einen schnelleren Algorithmus, den Bresenham-Algorithmus

```
/* Integer Bresenham-Algorithmus für den ersten Oktanten */
#include <Grafik-Grundlage.h>
   struct RastPort* rasp;

main()
{  int i;                               /* Alle verwendeten    */
                                        /* Variablen sind      */
   long x1,x2,y1,y2,x,y,dx,dy,e;        /* hier aufgeführt.    */
                                        /* Man beachte: keine  */
   Vorbereitung();                      /* Fließkommavariablen */
   rasp = window->RPort;
   x1 = 30;
   y1 = 30;              /* Koordinaten von P1 */
   x2 = 100;
   y2 = 80;              /* Koordinaten von P2 */

/* Beginn des Algorithmus */
   x = x1;
   y = y1;
   dx = x2 - x1;
   dy = y2 - y1;

/* Der neue Fehler e wird initialisiert */
   e =  2*dy -dx;

   for(i=1;i<=dx;i++)
      { WritePixel(rasp,x,y);
       while(e>=0)          /* Ab hier wurde das Programm */
         {                  /* dem neuen Fehler angepasst */
          y = y + 1;
          e = e - 2*dx;
         }
       x = x + 1;
       e = e + 2*dy;
      }

   Wait(1L<<window->UserPort->mp_SigBit);
   Ende();
}
```

© 1993 M&T

Listing 4: Nochmals der Bresenham-Algorithmus – diesmal arbeitet das Programm mit Integerzahlen

```
/* Integer Bresenham-Algorithmus für alle Oktanten */
#include <Grafik-Grundlage.h>
   struct RastPort* rasp;

main()
{  int i;                               /* Alle verwendeten    */
   int change,s1,s2;                    /* Variablen sind      */
   long x1,x2,y1,y2,x,y,dx,dy,e;        /* hier aufgeführt.    */

   Vorbereitung();
   rasp = window->RPort;
   x1 = 30;
   y1 = 30;              /* Koordinaten von P1 */
   x2 = 100;
   y2 = 80;              /* Koordinaten von P2 */
   x = x1;               /* Beginn des Algorithmus */
   y = y1;

   dx = abs( (float)(x2 - x1) );   /* Anzahl Pixel in x-Richtung */
   dy = abs( (float)(y2 - y1) );   /* Anzahl Pixel in y-Richtung */
   s1 = SPTst(x2 - x1);            /* Vorzeichen für x-Richtung  */
   s2 = SPTst(y2 - y1);            /* Vorzeichen für y-Richtung  */

   if(dy>dx)                       /* Entscheidung, in welcher   */
                                   /* Richtung Linie gezeichnet wird */
      { dy = dx;                      /* dx und dy werden */
        dx = abs( (float) (y2 - y1) );   /* vertauscht        */
        change = 1;            /* Flag für Vertauschung gesetzt */
      }
   else
      {change = 0;             /* Keine Vertauschung von dx und dx, */
      }                        /* da Flag nicht gesetzt ist */

/* Der Fehler e wird initialisiert */
   e =  2*dy - dx;
   for(i=1;i<=dx;i++)
      { WritePixel(rasp,x,y);

        while(e>=0)          /* Bei positivem e wird, je nachdem,   */
          {                  /* ob das Flag change gesetzt ist oder */
                             /* nicht, x bzw. y verändert.          */
           if(change==1)     /* Ob zu x bzw. y etwas addiert bzw.   */
             { x = x + s1;   /* subtrahiert wird, entscheidet das   */
             }               /* Vorzeichen s1 bzw. s2               */
           else
             { y = y + s2;
             }
           e = e - 2*dx;     /* Reinitialisierung von e             */
          }
       if(change==1)
         { y = y + s2;
         }
       else
         { x = x + s1;
         }
       e = e + 2*dy;
      }
   Wait(1L<<window->UserPort->mp_SigBit);

   Ende();
}
```

© 1993 M&T

Listing 5: Wir sind am Ziel: der komplette Integer-Bresenham-Algorithmus

IMPRESSUM

Stellv. Chefredakteur: Ulrich Brieden (ub) – verantwortlich für den redaktionellen Teil
Textchef: Jens Maasberg
Redaktion: Rainer Zeitler (rz)
freie Mitarbeiter: Ilse und Rudolf Wolf
Redaktionsassistenz: Catharina Winter

So erreichen Sie die Redaktion:
Tel. 0 89/46 13-4 14, Telefax: 0 89/46 13-4 33
Hotline Do, 15-17.oo Uhr

Manuskripteinsendungen: Manuskripte und Programmlistings werden gerne von der Redaktion angenommen. Sie müssen frei sein von Rechten Dritter. Sollten sie an anderer Stelle zur Veröffentlichung oder gewerblichen Nutzung angeboten worden sein, muß das angegeben werden. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck in der von Markt & Technik Verlag AG herausgegebenen Publikationen und zur Vervielfältigung der Programmlistings auf Datenträgern. Mit Einsendung von Bauanleitungen gibt der Einsender die Zustimmung zum Abdruck in von Markt & Technik Verlag AG verlegten Publikationen und dazu, daß die Markt & Technik Verlag AG Geräte und Bauteile nach der Anleitung herstellen läßt und vertreibt oder durch Dritte vertreiben läßt. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen.

Gestaltung und Desktop Publishing: Ulrich Brieden
Titel: Ulrich Brieden

Anzeigenleitung: Peter Kusterer
Anzeigenverwaltung und Disposition: Anja Böhl (233)

So erreichen Sie die Anzeigenabteilung:
Tel. 0 89/46 13-9 62, Telefax: 0 89/46 13-394

Leiter Vertriebsmarketing: Benno Gaab (740)

Vertrieb Handel: MZV, Moderner Zeitschriftenvertrieb GmbH & Co KG, Breslauer Straße 5, Postfach 11 23, 85386 Eching, Tel. 0 89/31 90 06-0

Erscheinungsweise: Das Sonderheft »Faszination Programmieren« ist zunächst zweimal jährlich geplant; die Erstausgabe kann noch nachbestellt werden

Bezugspreise: Das Einzelheft kostet DM 9,80 (Erstausgabe DM 12,-)

Leitung Technik: Wolfgang Meyer (887)

Druck: L.N.Schaffrath, Grafischer Betrieb, 47608 Geldern

Vorstand: Carl-Franz von Quadt (Vors.), Dr. Rainer Doll

Verlagsleitung: Wolfram Höfler
Produktionschef: Michael Koeppe

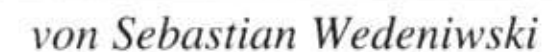
Direktor Zeitschriften: Michael M. Pauly

Anschrift des Verlages: Markt & Technik Verlag Aktiengesellschaft, Postfach 1304, 85531 Haar bei München, Telefon 089/4613-0, Telex 522052, Telefax 089/4613-100

Mathematische Algorithmen

Wurzel ziehen

Wie berechnet der Computer die Wurzel einer Zahl? Wir stellen Ihnen hier einen Algorithmus vor, und setzen ihn in Assembler um. So erhalten Sie eine besonders schnelle Routine, die Sie nach Belieben in eigenen Programmen einsetzen können, bei denen es auf Tempo ankommt.



von Sebastian Wedeniwski

Kopfrechnen ist out! Sobald man eine kompliziertere Formel berechnen muß, nehmen die meisten lieber einen Taschenrechner zur Hand und lösen so ihre Probleme. Und so geraten die eigentlichen mathematischen Grundlagen immer mehr in Vergessenheit. Die Wurzel ist einfach die Taste rechts oben auf dem kleinen Rechenmonster.

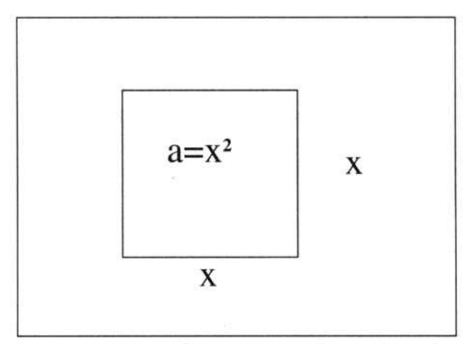
Doch nicht mit uns: Wir zeigen Ihnen hier einmal, wie man eine Funktion wie die Wurzel-Funktion auf dem Computer umsetzt, d.h. wir beleuchten einige Algorithmen und implementieren das Ganze in Assembler.

Annäherungstaktik zum Wurzelziehen

Die Aufgabe sieht wie folgt aus: Für eine positive Zahl a soll die Zahl $\sqrt{a}$ (= SQR(a)) näherungsweise ermittelt werden.

Bevor wir auf die exakten Verfahren eingehen, werfen Sie einen Blick auf den Kasten unten mit dem Titel »Annäherung«. Ein Beispiel, das zeigt, wie man die Wurzel durch Probieren im Kopf ausrechnen kann. Das Ganze ist sicher recht simpel und nicht sehr zielstrebig; es zeigt aber genau, wie man durch langsames »Annähern« den tatsächlichen Wert immer exakter eingrenzt. Die mathematischen Verfahren arbeiten nun ähnlich, sind allerdings um einiges effektiver:

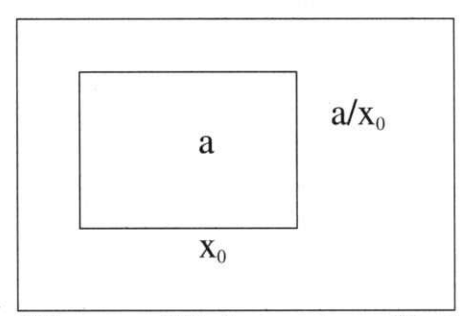
❐ Das erste Verfahren auf das wir eingehen wollen, geht auf Heron von Alexandria zurück (um 60 n. Chr.). Die Methode verfeinert den Weg, den wir am Beispiel im Kopf vollzogen haben. Das Verfahren wird durch eine geometrische Interpretation verdeutlicht:



Uralt: Das Verfahren von Heron beruht auf der quadratischen Gleichung

Die Seitenlänge x eines Quadrats vom Flächeninhalt a wird gesucht. Mit dem ersten Näherungswert $\mathbf{x_0}$ wird begonnen ($\mathbf{x_0}$ ist entweder zu groß oder zu klein). Wenn $\mathbf{x_0}$ die Länge eines Rechtecks ist, das denselben Flächeninhalt **a** wie das vorgegebene Quadrat hat – muß die Breite $\mathbf{a/x_0}$ sein. Der nächste Schritt wäre es, einen besseren Näherungswert $\mathbf{x_1}$ zu ermitteln: Wenn $\mathbf{x_0}$ zu groß ist, ist $\mathbf{a/x_0}$ zu klein und umgekehrt. Folglich wird der Umfang **U** des Rechtecks ermittelt:

$$U= 2x_0+2a/x_0 \leftrightarrow U/4=1/2*(x_0+a/x_0)$$



Näherung: Mit dem ersten x-Wert bekommen wir noch ein Rechteck

Den Umfang **U** durch vier geteilt liefert als arithmetisches Mittel von $\mathbf{x_0}$ und $\mathbf{a/x_0}$ einen besseren Näherungswert $\mathbf{x_1}$. Dieser Prozeß kann weiter fortgesetzt werden, d.h. der Wert $\mathbf{x_1}$ wird als neuer $\mathbf{x_0}$-Wert für eine weitere Berechnung eingesetzt und liefert dann einen Wert $\mathbf{x_2}$ etc.

Der neue Wert $\mathbf{x_{n+1}}$ entsteht dadurch, daß jeweils das arithmetische Mittel aus dem alten Näherungswert $\mathbf{x_n}$ und dem Quotienten $\mathbf{a/x_n}$ gebildet wird. Die explizite Formel lautet:

$$x_{n+1}= 1/2*(x_n+a/x_n)$$

❐ Ein weiteres Verfahren, das kurz erwähnt werden soll, ist das Näherungsverfahren von Newton (Nullstellenbestimmung einer Funktion): Die Tangente im Punkt **(x,f(x))** hat die Gleichung:

$$y = f(x_n)+f'(x_n)(x-x_n) \quad ; \text{ Nullstelle } y = 0$$
$$\leftrightarrow x_{n+1} = x_n-f(x_n)/f'(x_n)$$

Um nun die Wurzel von a zu bestimmen, wird die Funktion benötigt, deren Nullstelle das gesuchte Ergebnis ist:

$$f(x) = x^2-a \;;\; f'(x) = 2x$$

❐ Man kann die Quadratwurzel auch schriftlich berechnen. Zunächst muß eine reele Zahl **x** von rechts nach links in Gruppen zu je zwei Ziffern zerlegt werden. Die fundamentale Formel, um die Quadratwurzel zu berechnen, lautet:

$$(a+b+c+....)^2=a^2+(2a+b)b+2a+2b+c)c+...$$

Andere Schreibweise:

$$c\varepsilon N;\; d\varepsilon\{0..9\} \text{ (Ziffer)}$$
$$(10c+d)^2 = 100c^2+(20c+d)d$$

Der erste Schritt sieht so aus, daß aus der am weitesten links stehenden Zweiergruppe die größte Quadratzahl ($\mathbf{c^2}$), die kleiner oder gleich dieser Gruppe ist, bestimmt wird, wobei dieses c die erste Ziffer der gesuchten Quadratwurzel darstellt.

Ab jetzt bekommt dieses Verfahren ähnliche Strukturen des schriftlichen Dividierens, indem die Quadratwurzel von der Zweierwurzel abgezogen wird und das Ergebnis **(e)** darunter geschrieben wird. Zuersteinmal wird von dieser Zahl **e** die letzte Ziffer weggelassen und durch **2c** dividiert. So erhält man die nächste Ziffer

Annäherung

Wie kann man die Wurzel einer Zahl im Kopf berechnen? Am besten durch schrittweises Nähertasten ans gesuchte Ergebnis, wie folgendes Beispiel zeigt:
Nehmen wir die Wurzel aus 8? Wir wissen, die Wurzel aus 9 ist 3 und aus 4 ist es 2. Die gesuchte Wurzel muß irgendwo dazwischenliegen.
Versuchen wir's mit 2,5: Dann müßte 2,5 multipliziert mit sich selbst wieder die Ausgangszahl 8 ergeben. Allerdings ergibt 2,5 * 2,5 nur 6,25, d.h. unsere Wurzel ist um einiges zu klein.
Besser wäre ein Wert von 2,9. Wir rechnen 2,9 * 2,9... das gäbe 8,41... zu groß. Wir müssen unser vorläufiges Ergebnis von 2,9 etwas kleiner wählen... 2,8. Jetzt kommen wir auf 2,8*2,8 = 7,84. Wieder zu klein – aber nicht viel. Nehmen wir jetzt 2,83...usw.

d der Quadratwurzel, und es wird **d(20c+d)** von **e** abgezogen.

Ist aber das Produkt größer als die erweiterte Differenz, ist der Divisor durch die passende kleine Ziffer zu erweitern. Nun wird **c** durch **10c+d** ersetzt und der Algorithmus solange wiederholt, bis die gewünschte Anzahl Ziffern berechnet ist. Stößt man ans Komma, wird auch im Ergebnis an dieser Stelle das Komma geschrieben. Ein Beispiel:

```
√2|34|,|56|00|00|..-.=) 15,315 (c=1)
-1
 134              (13/2 : d=5)
-125              (25*5 : e = 125)
   956            (95/30 : d=3)
  -909            (303*3 : e=909)
    4700          (470/306 : d=1)
   -3061          (3061*1 : e=3061)
     163900       (16390/3062 : d=5)
    -153125       (30625*5 : e=153125)
      10875
```

❐ Anhand des schriftlichen Wurzelziehens gibt es ein Verfahren, das durch eine Modifikation für den Computer besonders geeignet ist.

Zunächst einige Definitionen:

$c \in N$ **(Resultat;**
$r \in N$ **(Radikant);**
$d \in \{0..9\}$ **(Ziffer)**

Den ganzen Anteil einer reelen Zahl **x** bezeichnet man $\lfloor x \rfloor$, z.B. $\lfloor 1{,}42 \rfloor = 1$

$x = \lfloor \sqrt{r/10^k} \rfloor * 10^k$ bedeutet, daß **x** dasjenige Vielfache von 10^k ist, für das $r-x^2$ seinen kleinsten nichtnegativen Wert annimmt, z.B.:

$$r = 3457 \quad \lfloor \sqrt{r/10} \rfloor * 10 = 50$$

denn $r-50^2 = 957$, aber $r-60^2 = -143<0$

Der Ausdruck $r-x^2$ kann sogar durch $\lfloor r/10^{2k} \rfloor * 10^{2k} - x^2$ ersetzt werden. Im Beispiel würde dies ergeben:

$$3400-50^2 = 900; 3400-60^2 = -200<0$$

$\lfloor \sqrt{r/10^{k+1}} \rfloor = c_{k+1}$;
$\lfloor \sqrt{r/10^{k}} \rfloor = c_{k}$;
$\lfloor r/10^{2k} \rfloor = r_{k}$;

Für die k-te Dezimalstelle d_k gilt:

$$c_k = 10 * c_{k+1} + d_k$$

Nach der Folgerung muß d_k so gewählt werden, daß $r_k * 10^{2k} - (c_k)^2 * 10^{2k}$ nichtnegativ, aber so klein wie möglich wird; und es wird minimal, wenn $r_k - (c_k)^2$ minimal wird. Das ist aber gleich

$$r_k-(10c_{k+1} + d_k)^2 =$$
$$r_k-100(c_{k+1})^2-20c_{k+1}d_k-((d_k)^2 =$$
$$(r_k-(10c_{k+1})^2)-d_k(2*10c_{k+1}+d_k).$$

Bekannt sind r_k und c_{k+1} und gesucht wird dann nur noch ein passender Wert für d_k.

Im Beispiel:

$r_0 = 3457$; $c_1 = 5$; $957-d_0(100+d_0)$;
$d_0=8$, da $8*(100+8) = 864$, aber
$9*(100+9)=981>957$

Durch wiederholte Anwendung dieses Verfahrens können der Reihe nach alle Dezimalstellen von $\sqrt{(r)}$ bestimmt werden. Allerdings muß man in jedem Schritt den obigen Ausdruck minimieren und auf nichtnegativ überprüfen.

Durch Verändern der Basis des Zahlensystems von bisher 10 in 2 (Binärsystem) bleiben alle Überlegungen trotzdem richtig und zusätzlich ergibt sich eine wesentliche Vereinfachung, nämlich für dk gibt es nur zwei Möglichkeiten.

Der bei der Wahl von d_k zu minimierende Ausdruck ist nun:

$$d_k \in \{0,1\}; (r_k-(2c_{k+1})^2)-d_k(2*2c_{k+1}+d_k)$$

Deswegen genügt ein einfacher Vergleich:
falls $r_k-4(c_{k+1})^2 > 4c_{k+1}$ ist, dann $d_k=0$
falls $r_k-4(c_{k+1})^2 => 4c_{k+1}$ ist, dann $d_k=1$

Die dabei auftretende Multiplikation mit 4 entspricht einer einfachen Verschiebung um zwei Bit nach links.

Nun haben Sie zwei verschiedene Verfahren zur Berechnung der Quadratwurzel kennengelernt – wo liegen die Vor- und Nachteile?

Das erste Verfahren, eine iterative Berechnung von $\sqrt{(a)}$, hat eine wichtige Eigenschaft: die Folge der Näherungswerte x_m konvergiert quadratisch gegen $\sqrt{(a)}$, d.h. wenn die erste k Ziffern von x_m mir $\sqrt{(a)}$ übereinstimmen, stimmt bei x_{m+1} mindestens 2k Ziffern mit der Zahlenfolge von $\sqrt{(a)}$ überein. Im zweiten Verfahren (Newton) trifft man die gleiche quadratische konvergierende Eigenschaft. Beim letzten Verfahren konvergiert es linear, aber ohne Division, die in Assembler viel Zeit beansprucht. Dieses Verfahren ist durch eine Modifikation für sehr lange Zahlen besser geeignet, da nur Verschiebungen und Subtraktionen benötigt werden. Doch nun konkret zur Implementation »Wurzel.asm«.

Wurzel in Assembler

Der Radikant (32 Bit) steht im Datenregister D0 und das Resultat (16 Bit) erhält man im Datenregister D1. Man muß einen der erwähnten Algorithmen (bin_wurzel, bin_wurzel2, Heron oder Newton) mit bsr/jsr aufrufen, um das Resultat zu erhalten. Die Algorithmen bin_wurzel und bin_wurzel2 sind weitgehend identisch, nur daß die zweite Version speziell auch für die Verwendung von längeren Radikanten modifiziert wurde (besteht nur noch aus Verschieben, Vergleichen und Subtrahieren; Orientierungshilfe: x-Flag). Die zwei Implementationen sind im Langwortbereich um Vielfaches schneller als die anderen.

Versuchen Sie, die einzelnen Implementationen auszubauen und evtl. zu optimieren. Finden Sie einen Trick, Wurzeln noch schneller zu ziehen? Und wenn Sie ein besonders schnelles Programm anzubieten haben, bitte sehr! Schicken Sie Ihre Lösung an die Redaktion und vielleicht sind Sie im nächsten »Faszination Programmieren« dabei.

Zum Schluß seien noch ein paar andere Möglichkeiten des Wurzelziehens genannt, die evtl . noch besser zu implementieren sind: Pellsche Gleichung, Kettenbrüche, euklidscher Algorithmus und Fionacci-Zahlen. Auch hier sollten Sie ein wenig experimentieren. ■

```
; Wurzel.asm berechnet Wurzeln

bit_length  =   32
rad         =   200000000

  move.l     #rad,d0
  bsr        bin_wurzel
  ;bsr       bin_wurzel2
  ;bsr       Heron
  ;bsr       Newton
  rts
bin_wurzel:
  move.l     #$40000000,d1
  moveq      #bit_length-3,d7
start:
  bset       d7,d1
  lsr.l      #1,d1
  bchg       d7,d1
  cmp.l      d1,d0
  bmi.s      loop
  sub.l      d1,d0
  bset       d7,d1
loop:
  subq       #2,d7
  bpl.s      start
  lsr        #1,d1
  rts
bin_wurzel2:
  move.l     #$40000000,d1
  move.l     #$30000000,d7
start2:
  lsr.l      #1,d1
  eor.l      d7,d1
  cmp.l      d1,d0
  bmi.s      loop2
  sub.l      d1,d0
  or.l       d7,d1
loop2:
  lsr.l      #2,d7
  bcc.s      start2
  lsr        #1,d1
  rts
Heron:
  moveq      #0,d3
  subq       #1,d3
  moveq      #1,d1
heron_1:
  move.l     d0,d2
  divu       d1,d2
  and.l      d3,d2
  add.l      d2,d1
  lsr.l      #1,d1
  cmp        d2,d1
  bne.s      heron_1
  rts

Newton:
  moveq      #0,d3
  moveq      #1,d1
newt_1:
  add        d3,d1
  move       d1,d2
  mulu       d2,d2
  move.l     d0,d3
  sub.l      d2,d3
  divs       d1,d3
  asr        #1,d3
  bne.s      newt_1
  rts           ;
```


»Wurzel.asm«: Ein Assembler-Programm, das Wurzeln berechnet

Multitasking: Tasks und Signale

Scheibchenweise

Eines hat der Amiga vielen Konkurrenten voraus: Multitasking. Wie das funktioniert, welche Datenstrukturen für den Programmierer relevant sind und welche Synchronisationsmechanismen zur Verfügung stehen – hier erfahren Sie's.

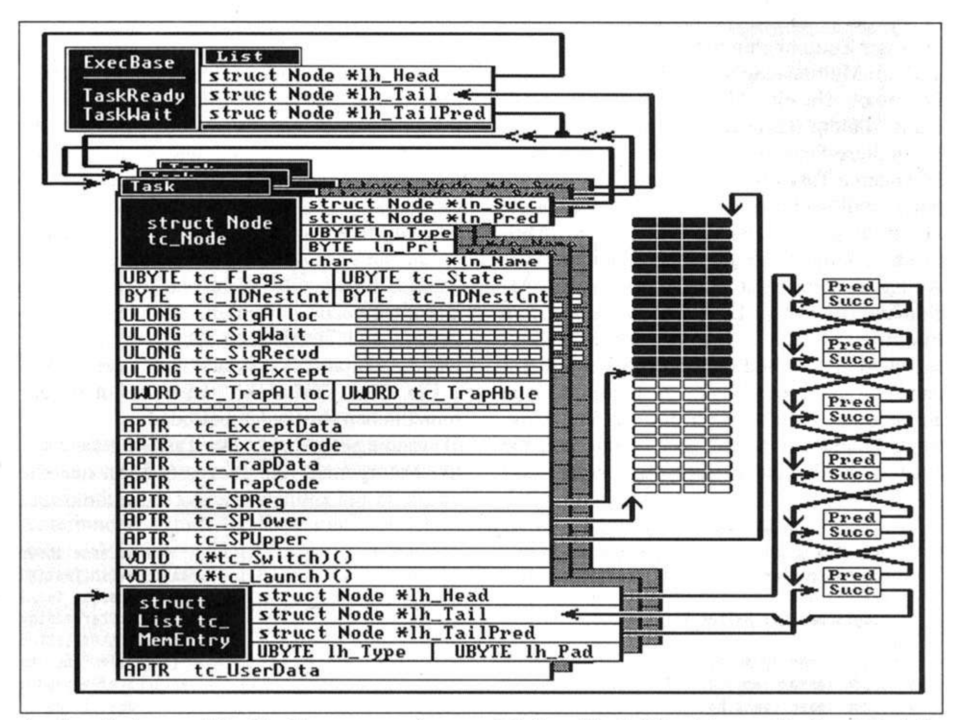
von Franz-Josef Reichert

Was ist überhaupt »Multitasking«? Im Grunde drückt dieser Begriff nur aus, daß mehrere Programme zur gleichen Zeit im Hauptspeicher des Amiga vorhanden sind und abwechselnd ausgeführt werden können: Sie teilen sich untereinander einen einzigen Prozessor. Das alleine ist allerdings nicht das eigentlich Bemerkenswerte. Interessanter ist das »Wie«: Wie findet diese Aufteilung statt? Von der verwendeten Technik hängt entscheidend die Leistungsfähigkeit eines Multitasking-Systems ab.

Die einfachste Strategie wäre, die verfügbare Prozessorzeit gleichmäßig auf alle konkurrierenden Programme zu verteilen. Zwar wäre das auf den ersten Blick legitim, aber nur in den wenigsten Fällen sinnvoll. Da verschiedene Programme unterschiedliche Bedürfnisse bezüglich ihrer Prozessornutzung haben, würde dies ständig zu ungenutzten Überkapazitäten für die eine Gruppe, chronischer Unterversorgung hingegen der anderen Programmgruppen führen. Ein weiterführender Ansatz ist daher die bedarfsorientierte Verteilung der knappen Ressource Prozessorzeit. Hierbei wird gefordert, daß Wartezeiten eines Programms, wie sie etwa bei IO-Operationen auftreten, den anderen Programmen zugute kommen sollen. Darüber hinaus existiert noch eine zusätzliche Strategie: Die sog. priorisierte Verteilung. Dabei erhalten weniger wichtige Programme entsprechend seltener als Anwendungen mit höherer Priorität die Chance, Prozessorzeit für sich zu beanspruchen.

Der Amiga beherrscht »real time, message-based multitasking« [1]. »real-time«, weil der Amiga in der Lage ist, auf Ereignisse in Echtzeit zu reagieren. Vom Standpunkt des Benutzers hat es den Anschein, daß nur dieses eine Programm bearbeitet wird. »Message-based«, weil der Datenaustausch und damit auch die Multitaskingsteuerung grundsätzlich über ein Nachrichtensystem erfolgen.

Programme bezeichnet man als Tasks. Jedem Task wird eine bestimmte Priorität zugeordnet. Über die bekannte »Node«-Struktur, die außerdem der Verkettung von Systemlisten dient, wird jedem Task eine Wertigkeit im Bereich von -128 bis +127 zugeteilt. Je höher die Wertigkeit, umso mehr Prozessorzeit fällt dem Task zu. Der »Task-Scheduler«, ebenfalls ein Teil des Exec, ist nun dafür verantwortlich, daß jeder Task zum Zuge kommt, also gelegentlich



Amiga-Interna: Die für Programmierer wichtige Task-Struktur mit allen Einträgen im Überblick

für eine kurze Zeitspanne den Prozessor beanspruchen kann. Exec aktiviert die Tasks über sein Interrupt-System in regelmäßigen Abständen, im Durchschnitt etwa zehnmal pro Sekunde. Normalerweise kann jeder Task nahezu zu jedem beliebigen Zeitpunkt Prozessorzeit erhalten oder den Zugriff darauf verlieren: Man spricht daher von »zuvorkommender Zeitplanung« oder »preemptive scheduling«. Bei einem Taskwechsel werden alle Prozessorregister des alten Tasks auf seinen Stack gerettet und die des neuen vom Stack geladen. Der neue Task setzt also seine Abarbeitung genau an der Stelle fort, wo ihn der Scheduler das letzte Mal unterbrochen hat. Alle Tasks, die gerade keine Prozessorzeit haben, befinden sich in einer Warteschlange. Ein Wechsel des aktiven Tasks findet genau dann statt, wenn eines der folgenden Ereignisse eintritt:

- Ein Task mit höherer Priorität wird der Warteschlange zugeführt, der dem gerade aktiven Task zuvorkommt.
- Der aktive Task muß auf ein externes Ereignis warten und daher den Prozessor von sich aus abgeben.
- Der aktive Task hat die ihm zugeteilte Zeitspanne verbraucht und ein anderer mit gleicher Priorität steht bereit, Kontrolle über den Prozessor zu erlangen (Zeitscheibenverteilung, »time-slicing«).

Mit seinem »preemptiven« Multitasking verfügt das Amiga-Betriebssystem über ein recht ökonomisches Verteilungsschema. Im Falle eines einzelnen Tasks ohne Konkurrenz erhält dieser die verfügbare Prozessorzeit und kann sich ungestört seinen Aufgaben widmen. Kommt ein Task niederer Priorität hinzu, bekommt dieser genau dann Prozessorzeit, wenn alle Tasks mit höherer Priorität selbst gerade keine benötigen. Tasks gleicher Prioritätsstufen teilen dagegen brüderlich die Prozessorzeit miteinander.

Warten – aber richtig:

Wir sprachen es schon des öfteren an: Das »Warten auf ein externes Ereignis« als steuerndes Element des Multitaskings. Doch was ist das eigentlich? Mit dem Übergang batch-orientierter (Stapelverarbeitung) zu interaktiven (dialogorientierten) Betriebssystemen stellte sich auch an sämtliche Programme die Anforderung, interaktiv auf Benutzereingaben zu reagieren, während früher einfach nur stur Rechenvorschriften mit vorgegebenen Daten ab-

gearbeitet und am Ende das Ergebnis ausgegeben wurde. Beim Amiga ist das anders. Hier liegt das Augenmerk auf der Kommunikation zwischen Programm und Anwender: dem Dialog. Muß ein Programm im Multitasking-Betriebssystem warten, harrt es z.B. auf die Tastatureingabe des Benutzers aus. In einem Singletask-System ist das ohne weitere Anforderungen durchführbar: Eine Endlosschleife fragt die Tastatur ab, bis das gewünschte Ereignis eintritt. Das ist zwar keine besonders sinnvolle Beschäftigung für den Prozessor, aber es schadet ja auch nicht, da für den Computer in dieser Zeit ohnehin nichts zu tun ist.

Beim Multitasking hingegen bedeutet es aus Prozessorsicht eine höllische Zeitverschwendung, ständig irgendein Ereignis abzufragen. Denn diese Prozessorzeit könnte viel sinnvoller anderen Tasks zugeführt werden. Das Warten in Endlosschleifen, auch »Busy Wait« oder »Polling« genannt, ist also Gift für jede Multitasking-Umgebung und daher nicht nur beim Amiga zu vermeiden. In der modularen Architektur des Amiga-Betriebssystems ist jedem Funktionsbereich seine eigene Softwareschnittstelle zugeordnet, und für Tastatureingaben ist das Input-Device zuständig. Salopp ausgedrückt gestaltet sich das »multitaskingfreundliche Warten« auf eine Tastatureingabe etwa folgendermaßen: Der Task signalisiert Exec, daß er auf ein Ereignis warten muß. Dies führt dazu, daß er zunächst aus der Warteschlange der um Prozessorzeit konkurrierenden Tasks ausgekoppelt wird und nun gar nichts mehr tut, bis ein Signal vom Input-Device eintrifft. Anhand des gesetzten Signales erkennt Exec, daß der Wartezustand des Tasks aufgehoben ist und klinkt ihn wieder in die Warteschlange ein. Wenn der Task das nächste Mal Prozessorzeit erhält, kann er die Nachricht des Input-Devices, also die Tastatureingabe, auswerten. Das Wesentliche: Im Wartezustand benötigt der Task keinerlei Prozessor-Ressourcen.

Signale sind ein grundlegender Mechanismus von Exec, der die Basis für alle Nachrichtensysteme, Synchronisationsaufgaben und Zugriffsprotokolle darstellt und damit eine entscheidende Voraussetzung zum reibungslosen Multitasking schafft. Der gleiche Mechanismus gilt natürlich auch für alle weiteren Ereignisse, die auf dem Amiga im wesentlich komplexeren Umfang stattfinden: Mausbewegungen, Diskettenwechsel, Schnittstellenoperationen, Grafik- und Tonausgabe zählen ebenso dazu wie die simple Textausgabe in ein Terminal.

Ein Task kann sich demnach in unterschiedlichen Zuständen befinden:

❒ »running« (laufend): Der Task ist gerade aktiv. Naturgemäß (ein Prozessor kann nur ein Programm abarbeiten) kann dies immer nur ein Task, niemals mehrere gleichzeitig, sein. Wird der Prozessor nicht gerade zur Bearbeitung von Ausnahmebedingungen (Interrupts, Traps und Exceptions) benötigt, kann der Task frei über ihn verfügen.

❒ »ready« (bereit): Der Task befindet sich in der Bereitschaft, bei nächster Gelegenheit CPU-Zeit zugeteilt zu bekommen: Er konkurriert also mit anderen Tasks im »time-slicing« (Zeitscheibenverteilung) um Prozessorzeit.

❒ »waiting« (wartend): Der Task beansprucht momentan keine CPU-Zeit, da er auf das Eintreffen eines externen Ereignisses warten muß. Sobald es eintrifft, ändert sich sein Zustand in »ready«.

Die Task-Struktur:

Schauen wir uns nun die Struktur des »Task-Control-Blocks« an (Bild »Amiga-Interna« erste Seite dieses Artikels). Exec verwaltet eine private »TaskReady«-Liste für alle Tasks im »ready«-Status, und eine private »TaskWait«-Liste für alle sich im »waiting«-Status befindlichen Tasks. Zum Auffinden eines Tasks mit bekanntem Namen existiert die Funktion »FindTask()«. Sie durchforstet beide Listen und retourniert die Task-Adresse. Übergibt man Null, läßt sich so die Adresse des eigenen Tasks in Erfahrung bringen.

```
/* Dieses Programm demonstriert die Task-Generation
 * und eine mögliche Kommunikationsfähigkeit.
 * Beim Kompilieren unbedingt die StackCheck-Option
 * deaktivieren.
 * Programmautor: Rainer Zeitler
 */
#include <exec/types.h>
#include <exec/memory.h>
#include <exec/tasks.h>
#include <proto/dos.h>
#include <proto/exec.h>
#include <stdio.h>

#define TASKPRI   0L
#define TASKNAME  "TochterTask"
#define STACKSIZE 1000L
/* Darüber läßt sich nachvollziehen, daß der
 * Tochtertask überhaupt seine Arbeit verrichtete     */
ULONG TochterCounter=0;
/* Signal-Bits für die Kommunikation */
ULONG TochterStartedSig=0,StopTochterTaskSig=0;
/* Der Task des Hauptprogramms */
struct Task *MainTask;
void __saveds __interrupt TochterTask(void) {
  /* Das Stopsignal anlegen */
  StopTochterTaskSig=AllocSignal(-1L);
  /* Dem Haupttask mitteilen, daß wir soweit sind */
  Signal( MainTask, 1L<<TochterStartedSig );
  /* Solange die Variable hochzählen, bis das Stop-
   * Signal geschickt wird       */
  while( (SetSignal(0L,0L) &
                (1L<<StopTochterTaskSig)) == 0)
    TochterCounter++;
  /* Signal wieder freigeben */
  FreeSignal( StopTochterTaskSig );
  /* Beenden bestätigen */
  Signal( MainTask, 1L<<TochterStartedSig );
  /* Wichtig: So schläft der Task für immer und
   * wir können ihn gefahrlos via RemTask() entfernen     */
  Wait(0);
}
main(long argc, char **argv) {
  struct Task *Tochter=AllocMem(sizeof(struct Task),
                          MEMF_CLEAR|MEMF_PUBLIC);
  APTR   TaskStack=AllocMem(STACKSIZE,
                          MEMF_CLEAR|MEMF_PUBLIC);
  /* Die Adresse unseres Tasks */
  MainTask=FindTask(NULL);
  if( Tochter && TaskStack )
  { TochterStartedSig=AllocSignal(-1L);
    if( TochterStartedSig )
    { /* Priorität des Tochter-Tasks */
      Tochter->tc_Node.ln_Pri = TASKPRI;
      /* Jawoll, es ist ein Task */
      Tochter->tc_Node.ln_Type = NT_TASK;
      /* Der Name des Tochter-Tasks */
      Tochter->tc_Node.ln_Name = TASKNAME;
      /* Untere Grenze des Tochter-Task-Stacks */
      Tochter->tc_SPLower = TaskStack;
      /* Obere Grenze des Tochter-Task-Stacks */
      Tochter->tc_SPUpper = Tochter->tc_SPReg =
            (APTR)((ULONG)TaskStack+STACKSIZE);
      /* Task starten */
      AddTask(Tochter,(APTR)TochterTask,0L);
      /* Warten, bis der TochterTask sich meldet */
      printf("Warte auf TochterTask\n");
      Wait( 1L<<TochterStartedSig );
      printf("TochterTask hat sich gemeldet\n" "5 Sekunden warten\n");
      Delay( 5 * 50 );
      /* Den TochterTask beenden */
      printf("Beende TochterTask und warte" " auf Bestätigung\n");
      Signal( Tochter, 1L<<StopTochterTaskSig );
      /* Auf Bestätigung warten */
      Wait( 1L<<TochterStartedSig );
      printf("TochterTask beendet\n");
      printf("Zählerstand: %ld\n",TochterCounter);
      /* Task entfernen */
      RemTask( Tochter );
      /* Signal freigeben */
      FreeSignal( TochterStartedSig );
  } }
  /* Speicher freigeben */
  if( TaskStack )
    FreeMem( TaskStack, STACKSIZE );
  if( Tochter )
    FreeMem( Tochter, sizeof(struct Task) );
}
```


»TaskDemo.c«: Demonstriert das Einrichten eines Tasks und die Kommunikationsmöglichkeiten via Signalen

Ist ein Task nicht aktiv, wird er über »tc_Node« mit einer der beiden Listen verkettet. Der Eintrag »ln_Type« enthält den Wert »NT_TASK« (1), in »ln_Pri« läßt sich die Priorität angeben. Dies erfolgt in der Regel per Aufruf der Exec-Funktion »SetTaskPri()«. Werte im Bereich von -20 bis +20 stellen die Standardeinstellungen für System-Tasks dar. Besteht kein besonderer Anlaß für höhere Werte, sollte die Priorität eines Tasks grundsätzlich Null sein.

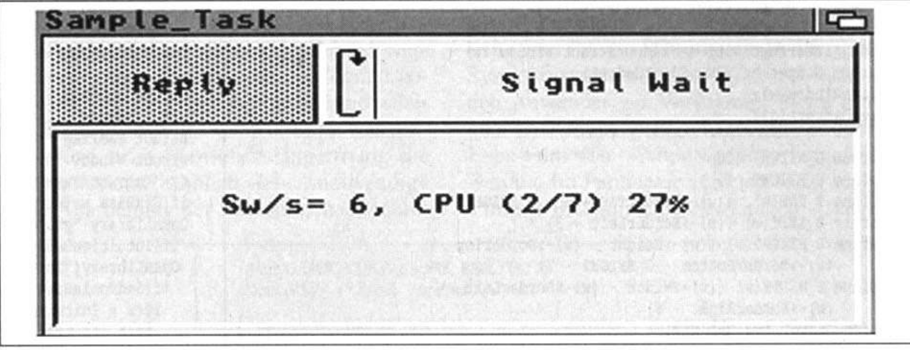
Gemäß den zuvor erläuterten Mechanismen des Multitaskings bringt es auch wenig, die Task-Priorität einfach wahllos zu erhöhen, etwa in der Hoffnung, das eigene Programm dadurch schneller zu machen. Wenn sich alle Tasks an die genannten Konventionen halten, erhält das prozessorintensivste Programm ohnehin die dickste Zeitscheibe. Unsinnig hohe Taskprioritäten auf Anwenderprogrammebene können allerdings zu ernsthaften Schwierigkeiten führen, wenn Systemprogramme deswegen nicht mehr genügend Prozessorzeit erhalten. Über den aktuellen Status des Tasks gibt der Eintrag »tc_State« Auskunft, wobei dieser ebenso wie »tc_Flags« von Exec verwaltet wird.

Multitasking-Steuerung:

»tc_IDNestCnt« steht für »Interrupt Disable Nesting Counter«, »tc_TDNestCnt« entsprechend für »Task Disable Nesting Counter«. Beides hängt mit wichtigen Kontrollmechanismen zusammen, über welche der Programmierer Einfluß auf die Multitaskingsteuerung nehmen kann.

Im Programmiermodell des MC680x0 existieren grundsätzlich zwei Zustände, in welchen System- und Anwenderprogramme abgearbeitet werden. Man unterscheidet zwischen Supervisor- und User-Modus. Der Supervisor-Modus ist den Interrupts, Prozessor-Exceptions und -Traps, also den Ausnahmebedingungen vorbehalten. Im User-Modus finden alle übrigen Programmabläufe statt. Nun kann es in manchen Fällen sinnvoll oder sogar notwendig sein, daß der gerade laufende Task nicht von anderen Tasks gestört oder unterbrochen wird, etwa beim Zugriff auf Systemstrukturen, die auch von anderen im User-Modus laufenden Tasks verändert werden können. Hierzu ist es notwendig, das Task-Scheduling kurzfristig zu unterbinden – mit der Exec-Funktion »Forbid()« (Verbieten). Das Pendant zum Wiedereintritt ins Multitasking lautet »Permit()« (Erlauben). Die Aufrufe sind schachtelbar, wobei jedesmal »tc_TDNestCnt« herauf- bzw. heruntergezählt wird. Solange dieser Wert ungleich Null ist, ist der aktuelle Task garantiert der einzige, der abgearbeitet wird. Existieren für teilbare Systemressourcen keine eigenen Zugriffsprotokolle und besteht die Gefahr, daß der Eingriff weiterer Benutzer zu Unbeständigkeiten führt, stellt dieser Mechanismus die letzte Möglichkeit einer Schutzmaßnahme dar.

Einen Schritt weiter kann man gehen, indem zusätzlich zum Task-Scheduling auch noch die im Supervisor-Modus laufenden Ausnahmebedingungen unterbunden werden. Dies ist beispielsweise nötig, wenn Systemstrukturen auch auf der Ebene der Ausnahmebehandlungen geändert werden müssen. Ein gutes Beispiel dafür sind die beiden erwähnten Systemlisten für Tasks. Der Task-Scheduler (auf Interrupt-Ebene) ist quasi zu jedem beliebigen Zeitpunkt am Werk. Wollte ein Task (im User-Modus) die Listen untersuchen – sie wären stets inkonsistent! Kaum hätte er den ersten Eintrag gelesen, könnte er schon wieder den Prozessor verlieren, und bei seiner nächsten Chance hät-



Tasks und Kommunikation: Mit diesem Requester meldet sich unser Demoprogramm und wartet auf Antwort

»TaskDemo_2.c«: Die zweite Demo ist noch ausführlicher und wartet auf Antworten des Benutzers (Listinganfang)

```
/* task example --- F.J. Reichert 1993 */
#include <exec/types.h>
#include <exec/memory.h>
#include <exec/tasks.h>
#include <devices/timer.h>
#include <graphics/gfxbase.h>
#include <graphics/gfx.h>
#include <graphics/gfxmacros.h>
#include <intuition/gadgetclass.h>
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <libraries/gadtools.h>
#include <proto/dos.h>
#include <proto/exec.h>
#include <proto/graphics.h>
#include <proto/timer.h>
#include <proto/intuition.h>
#include <proto/gadtools.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <clib/macros.h>
#define STACKSIZE  (4*1024)
#define TASKPRI  0
#define TASKNAME  "Sample_Task"
#define SIG_SHOWMSG 0
#define SIG_EXIT  1
#define SIG_REPLY  2
#define SIG_MAX  3
#define GAD_REPLY  0
#define GAD_MODE  1
#define GAD_MAX  2
struct MemTag {struct MemList mt_ML;
              struct MemEntry mt_ME;
};
#define TFLGF_TIMERSET 0x0001
#define TFLGF_WATCH  0x0002
#define TFLGF_BUSYWAIT 0x0004
struct TaskTag {
 struct Task tt_Task;
 ULONG tt_flags;
 UBYTE volatile tt_SigBit[SIG_MAX];
 struct Task* volatile tt_SigTask[SIG_MAX];
 UBYTE* volatile tt_message;
 UWORD volatile tt_msglen;
 ULONG volatile tt_lcount;
 struct MsgPort *tt_timerport;
 struct timerequest *tt_timerequest,tt_cyclic;
 struct Library *tt_timerbase;
 struct timeval volatile tt_tv0,tt_tvTskEntry,tt_tvUsed,tt_tvExpd;
 int volatile tt_cpu;
};
#define TSK_SHOWMSG(t) ((t)->tt_SigTask[SIG_SHOWMSG])
#define TSK_EXIT(t)    ((t)->tt_SigTask[SIG_EXIT])
#define TSK_REPLY(t)   ((t)->tt_SigTask[SIG_REPLY])
#define BIT_SHOWMSG(t) ((t)->tt_SigBit[SIG_SHOWMSG])
#define BIT_EXIT(t)    ((t)->tt_SigBit[SIG_EXIT])
#define BIT_REPLY(t)   ((t)->tt_SigBit[SIG_REPLY])
#define MSK_SHOWMSG(t) (1L << BIT_SHOWMSG(t))
#define MSK_EXIT(t)    (1L << BIT_EXIT(t))
#define MSK_REPLY(t)   (1L << BIT_REPLY(t))
#define TimerBase (t->tt_timerbase)
extern struct IntuitionBase *IntuitionBase;
extern struct Library *GadToolsBase;
extern struct GfxBase *GfxBase;
void __interrupt Launch(void) {
 struct TaskTag *t = (struct TaskTag*)FindTask(NULL);
 if(t->tt_flags & TFLGF_TIMERSET) {
  t->tt_flags |= TFLGF_WATCH;
  t->tt_lcount++;
```

```
  GetSysTime(&t->tt_tvTskEntry);
 }}
 #define MILLION 1000000L
 void __interrupt Switch(void) {
  struct TaskTag *t = (struct TaskTag*)FindTask(NULL);
  if(t->tt_flags & TFLGF_WATCH) {
   t->tt_flags &= ~TFLGF_WATCH;
   GetSysTime(&t->tt_timerequest->tr_time);
   t->tt_tvExpd = t->tt_timerequest->tr_time;
   SubTime(&t->tt_tvExpd,&t->tt_tv0);
   SubTime(&t->tt_timerequest->tr_time,&t->tt_tvTskEntry);
   AddTime(&t->tt_tvUsed,&t->tt_timerequest->tr_time);
   t->tt_cpu = 100L * (t->tt_tvUsed.tv_secs * MILLION
       + t->tt_tvUsed.tv_micro) / (t->tt_tvExpd.tv_secs
        * MILLION + t->tt_tvExpd.tv_micro);
 }}
 void __interrupt DisposeWindow(struct Window *w) {
  struct Gadget *g = w->FirstGadget;
  CloseWindow(w);
  FreeGadgets(g);
 }
 #define G_WIDTH  100
 #define G_HEIGHT  30
 #define B_TOP(w)  ((w)->BorderTop + 5 + G_HEIGHT)
 #define B_LEFT(w) ((w)->BorderLeft + 3)
 #define B_HEIGHT(w) ((w)->Height - (w)->BorderTop -\
        (w)->BorderBottom - G_HEIGHT - 7)
 #define B_WIDTH(w) ((w)->Width - (w)->BorderLeft -\
        (w)->BorderRight - 6)
 struct Gadget *wg[GAD_MAX];
 struct Window* __interrupt InitWindow(void) {
  APTR vi; struct NewGadget ng;
  struct Screen *ps; struct Window *w;
  struct Gadget *gl,*gad = NULL;
  if(ps = LockPubScreen(NULL)) {
   if(vi = GetVisualInfoA(ps,NULL)) {
    if(gad = CreateContext(&gl)) {
     static UBYTE *cycle_labels[] = {
      "Signal Wait","Busy Wait",NULL
     };
     ng.ng_LeftEdge  = ps->WBorLeft + 3;
     ng.ng_TopEdge  = ps->BarHeight + 4;
     ng.ng_Width    = G_WIDTH;
     ng.ng_Height   = G_HEIGHT;
     ng.ng_VisualInfo = vi;
     ng.ng_GadgetText = "Reply";
     ng.ng_TextAttr  = ps->Font;
     ng.ng_Flags    = PLACETEXT_IN;
     ng.ng_GadgetID  = 0;
     ng.ng_UserData  = NULL;
     wg[GAD_REPLY] = gad = CreateGadget(BUTTON_KIND,
     gad,&ng,GA_Disabled,1L,TAG_DONE);
     ng.ng_LeftEdge  = gad->LeftEdge + gad->Width + 1;
     ng.ng_Width    = G_WIDTH * 2 + 1;
     ng.ng_GadgetText = NULL;
     ng.ng_GadgetID  = 1;
     wg[GAD_MODE] = gad = CreateGadget(CYCLE_KIND,gad,&ng,
     GTCY_Labels,cycle_labels,
     GTCY_Active,0L,TAG_DONE);
     if(w = OpenWindowTags(NULL,
     WA_Flags,(ULONG)WFLG_DEPTHGADGET|WFLG_DRAGBAR,
     WA_IDCMP,(ULONG)IDCMP_GADGETUP,
     WA_InnerHeight,(ULONG)G_HEIGHT + 80L,
     WA_InnerWidth,(ULONG)G_WIDTH * 3 + 8,
     WA_Gadgets,gl,
     WA_Title,TASKNAME,
     WA_PubScreen,ps,
     TAG_DONE)) {
      GT_RefreshWindow(w,NULL);
      SetAPen(w->RPort,1);
      SetBPen(w->RPort,0);
      SetDrMd(w->RPort,JAM2);
      DrawBevelBox(w->RPort,B_LEFT(w),
      B_TOP(w),B_WIDTH(w),B_HEIGHT(w),
      GT_VisualInfo,vi,GTBB_Recessed,1L,TAG_DONE);
     }
     else FreeGadgets(gl);
    }
    FreeVisualInfo(vi);
   }
  UnlockPubScreen(NULL,ps);
  }
  return(w);
 }
```

»TaskDemo_2.c«: Demonstriert das Einrichten von Tasks und die Kommunikationsmöglichkeiten (Listingfortsetzung)

```
void __interrupt print(struct Window *w,UBYTE *text,int top,int len) {
 SetAPen(w->RPort,0);
 RectFill(w->RPort,B_LEFT(w) + 2,top -
 w->RPort->Font->tf_Baseline,B_WIDTH(w) + 4,
 top + w->RPort->Font->tf_YSize - w->RPort->Font->tf_Baseline);
 Move(w->RPort,B_LEFT(w) + (B_WIDTH(w) - TextLength
 (w->RPort,text,len)) / 2,top);
 SetAPen(w->RPort,1);
 Text(w->RPort,text,len);
}
void __interrupt ShowCPU(struct Window *w,struct TaskTag *t) {
 UBYTE cpu_string[32];
 sprintf(cpu_string,"Sw/s=%2ld, CPU (%ld/%ld) %2d%%",
 t->tt_lcount / MAX(t->tt_tvExpd.tv_secs,1),
 t->tt_tvUsed.tv_secs,t->tt_tvExpd.tv_secs,t->tt_cpu);
 print(w,cpu_string,B_HEIGHT(w) / 2 -
 w->RPort->Font->tf_Baseline + B_TOP(w),strlen(cpu_string));
}
#define AMIGAOS_V_37 37
void __saveds __interrupt SampleTask(void) {
 struct TaskTag *t;
 struct Window *w;
 t = (struct TaskTag*)FindTask(NULL);
 if(GfxBase = (struct GfxBase*)
 OpenLibrary("graphics.library",AMIGAOS_V_37)) {
  if(IntuitionBase = (struct IntuitionBase*)
  OpenLibrary("intuition.library",AMIGAOS_V_37)) {
   if(GadToolsBase = OpenLibrary("gadtools.library",AMIGAOS_V_37)) {
    if(w = InitWindow()) {
     if(t->tt_timerport = CreateMsgPort()) {
      if(t->tt_timerequest = (struct timerequest*)
      CreateIORequest(t->tt_timerport,sizeof(struct timerequest))) {
       if(OpenDevice(TIMERNAME,UNIT_VBLANK,
       t->tt_timerequest,0L) == 0) {
        t->tt_cyclic = *t->tt_timerequest;
        t->tt_timerbase = (struct Library*)
        t->tt_timerequest->tr_node.io_Device;
        GetSysTime(&t->tt_tv0);
        t->tt_flags |= TFLGF_TIMERSET;
        t->tt_cyclic.tr_node.io_Command = TR_ADDREQUEST;
        t->tt_cyclic.tr_time.tv_secs = 0L;
        t->tt_cyclic.tr_time.tv_micro = 400000L;
        SendIO(&t->tt_cyclic.tr_node);
        if((BIT_SHOWMSG(t) = AllocSignal(-1L)) != -1) {
         TSK_SHOWMSG(t) = FindTask(NULL);
         if((BIT_EXIT(t) = AllocSignal(-1L)) != -1) {
          TSK_EXIT(t) = FindTask(NULL);
          Signal(TSK_REPLY(t),MSK_REPLY(t));
          while(1) {
           ULONG sigset;
           if(t->tt_flags & TFLGF_BUSYWAIT) {
            sigset = SetSignal(0L,MSK_EXIT(t)
            | MSK_SHOWMSG(t)
            | 1L << t->tt_timerport->mp_SigBit
            | 1L << w->UserPort->mp_SigBit);
           }
           else {
            sigset = Wait(MSK_EXIT(t)
            | MSK_SHOWMSG(t)
            | 1L << t->tt_timerport->mp_SigBit
            | 1L << w->UserPort->mp_SigBit);
           }
           if(sigset & 1L << t->tt_timerport->mp_SigBit) {
            t->tt_cyclic.tr_node.io_Command = TR_ADDREQUEST;
            t->tt_cyclic.tr_time.tv_secs = 0L;
            t->tt_cyclic.tr_time.tv_micro = 400000L;
            SendIO(&t->tt_cyclic.tr_node);
            ShowCPU(w,t);
           }
           if(sigset & MSK_EXIT(t)) {
            break;
           }
           if(sigset & MSK_SHOWMSG(t)) {
            GT_SetGadgetAttrs(wg[GAD_REPLY],w,NULL,
            GA_Disabled,0L,TAG_DONE);
            RefreshGList(wg[0],w,NULL,1);
            print(w,t->tt_message,
            B_TOP(w) + B_HEIGHT(w) / 2 +
            w->RPort->Font->tf_Baseline,t->tt_msglen);
           }
           if(sigset & 1L << w->UserPort->mp_SigBit) {
            struct IntuiMessage *imsg;
            while(imsg = GT_GetIMsg(w->UserPort)) {
             ULONG im_code,im_class;
```

te der Scheduler möglicherweise die Liste schon wieder völlig umsortiert. Das hierfür vorgesehene Schutzprotokoll lautet »Disable()« (Unterbinden), das Gegenstück »Enable()« (Ermöglichen). Beide Funktionen schalten Ausnahmeverarbeitungen über das entsprechende Hardware-Register des Custom-Chips »Paula« aus oder ein. Die Zustandsvariable »tc_IDNestCnt« wird nach dem zuvor erläutertem Schema beeinflußt. Es ist im übrigen unsinnig, nach einem Disable() noch ein Forbid() nachzuschieben. Da der Task-Scheduler schon beim Ausschalten der Ausnahmeverarbeitungen stillgelegt wird, findet konsequenterweise auch kein Umschalten der Tasks statt.

Vitale Bedürfnisse

So nützlich die beschriebenen Mechanismen in bestimmten Fällen sein können, so gefährlich kann es werden, wenn allzu sorglos damit umgegangen wird. Auch wenn der Gebrauch von »Forbid()« auf den ersten Blick ungefährlich scheint, muß man sich doch stets vor Augen halten, daß jeder überflüssige Aufruf dem ansonsten gleichförmigen und störungsfreien Multitasking ein abruptes »Stillgestanden« verordnet. »Forbid()« ist also genauso wie das unüberlegte Heraufsetzen der Taskpriorität keineswegs ein probates Mittel, dem eigenen Programm zu mehr Geltung zu verhelfen, sondern unter dieser Prämisse schlicht ein nutzloser Störfaktor.

Richtig gefährlich kann dagegen sorgloses Unterbinden der Ausnahmebehandlungen werden. Das Amiga-Betriebssystem ist darauf angewiesen, daß diese peinlich genau und nahezu in Echtzeit ablaufen können. Denn was passiert, wenn beim Menschen längerfristig die Atmung aussetzt? Ähnlich »lebensnotwendig« ist die Abarbeitung der Ausnahmebehandlungen. Länger als 250 Mikrosekunden (0,00025 sec.) darf dieser Zustand nicht anhalten, sonst kann es passieren, daß sich der Amiga bis zum Enable()-Aufruf bereits verabschiedet hat. Das für den Benutzer sichtbare Resultat ist dann in der Regel ein Absturz.

Die vier in der Task-Struktur vorkommenden Einträge »tc_SigAlloc«, »tc_SigWait«, »tc_SigRecvd« und »tc_SigExcept« führen uns wieder zum erwähnten Signalisationsmechanismus für externe Ereignisse zurück. Insgesamt verfügt jeder Amiga-Task über 32 Signale – jedes Bit der genannten Langworteinträge repräsentiert eines davon. Die unteren 16 Bit sind für Systemzwecke reserviert, die oberen stehen für den Anwender zur Verfügung. Bevor ein Signal-Bit benutzt werden kann, muß es über die Exec-Funktion »AllocSignal()« angefordert werden. Im Parameter kann entweder eine bestimmte Bit-Nummer oder einfach -1 angege-

```
            APTR im_address;
            im_class = imsg->Class;
            im_code = imsg->Code;
            im_address = imsg->IAddress;
            GT_ReplyIMsg(imsg);
            switch(im_class) {
             case IDCMP_GADGETUP:
              switch(((struct Gadget*)
              im_address)->GadgetID) {
               case GAD_REPLY:
                GT_SetGadgetAttrs(wg[GAD_REPLY],
                w,NULL,GA_Disabled,1L,TAG_DONE);
                RefreshGList(wg[0],w,NULL,1);
                Signal(TSK_REPLY(t),MSK_REPLY(t));
                break;
               case GAD_MODE:
                switch(im_code) {
                 case 0:
                  t->tt_flags &= ~TFLGF_BUSYWAIT;
                  break;
                 case 1:
                  t->tt_flags |= TFLGF_BUSYWAIT;
                  break;
                 default:
                  break;
                }
                break;
               default:
                break;
              }
              break;
             default:
              break;
          }}}}
          AbortIO(&t->tt_cyclic.tr_node);
          WaitIO(&t->tt_cyclic.tr_node);
          TSK_EXIT(t) = NULL;
          FreeSignal(BIT_EXIT(t));
         }
         TSK_SHOWMSG(t) = NULL;
         FreeSignal(BIT_SHOWMSG(t));
        }
        t->tt_flags &= ~(TFLGF_TIMERSET|TFLGF_WATCH);
        CloseDevice(t->tt_timerequest);
       }
       DeleteIORequest(t->tt_timerequest);
      }
      DeleteMsgPort(t->tt_timerport);
     }
     DisposeWindow(w);
    }
    CloseLibrary(GadToolsBase);
   }
   CloseLibrary(&IntuitionBase->LibNode);
  }
  CloseLibrary(&GfxBase->LibNode);
 }
 Forbid();
  Signal(TSK_REPLY(t),MSK_REPLY(t));
  Wait(0L); /* Wait forever... */
 }
void main(void) {
 struct TaskTag *ptt; struct MemList *pml;
 struct MemTag aml; UBYTE SigBit; APTR stk;
 aml.mt_ML.ml_Node.ln_Type = NT_MEMORY;
 aml.mt_ML.ml_Node.ln_Pri  = 0;
 aml.mt_ML.ml_Node.ln_Name = TASKNAME;
 aml.mt_ML.ml_NumEntries   = 2;
 aml.mt_ML.ml_ME[0].me_Un.meu_Reqs = MEMF_PUBLIC|MEMF_CLEAR;
 aml.mt_ML.ml_ME[0].me_Length      = sizeof(struct TaskTag);
 aml.mt_ML.ml_ME[1].me_Un.meu_Reqs = MEMF_CLEAR;
 aml.mt_ML.ml_ME[1].me_Length      = STACKSIZE;
 pml = AllocEntry(&aml.mt_ML);
 if(!((ULONG)pml & (1L << 31))) {
  ptt = (struct TaskTag*) pml->ml_ME[0].me_Un.meu_Addr;
  stk = (APTR)            pml->ml_ME[1].me_Un.meu_Addr;;
  ptt->tt_Task.tc_Node.ln_Pri  = TASKPRI;
  ptt->tt_Task.tc_Node.ln_Type = NT_TASK;
  ptt->tt_Task.tc_Node.ln_Name = TASKNAME;
  ptt->tt_Task.tc_SPLower = stk;
  ptt->tt_Task.tc_SPReg   =
  ptt->tt_Task.tc_SPUpper = (APTR)((ULONG)stk + STACKSIZE);
  NewList(&ptt->tt_Task.tc_MemEntry);
  AddHead(&ptt->tt_Task.tc_MemEntry,&pml->ml_Node);
  if((SigBit = AllocSignal(-1L)) != -1) {
   BIT_REPLY(ptt) = SigBit;
   TSK_REPLY(ptt) = FindTask(NULL);
   AddTask(&ptt->tt_Task,(APTR)SampleTask,0L);
   Disable();
   ptt->tt_Task.tc_Launch  = Launch;
   ptt->tt_Task.tc_Switch  = Switch;
   ptt->tt_Task.tc_Flags  |= (TF_LAUNCH|TF_SWITCH);
   Enable();
   Wait(MSK_REPLY(ptt));
   while(1) {
    int len;
    UBYTE msg_buf[128];
    Write(Output(),"Parent: ",8);
    len = Read(Input(),msg_buf,128);
    if(*msg_buf == '\n') break;
    ptt->tt_message = msg_buf;
    ptt->tt_msglen = len - 1;
    Signal(TSK_SHOWMSG(ptt),MSK_SHOWMSG(ptt));
    Wait(MSK_REPLY(ptt));
   }
   Signal(TSK_EXIT(ptt),MSK_EXIT(ptt));
   Wait(MSK_REPLY(ptt));
   RemTask(&ptt->tt_Task);
   FreeSignal(SigBit);
  }
 }
 exit(0);
}      © 1993 M&T
```

»TaskDemo_2.c«: Demonstriert das Einrichten von Tasks und die Kommunikationsmöglichkeiten (Listingende)

ben werden, womit die nächste freie Signal-Bit-Nummer angefordert wird. Die Funktion liefert entweder eine Bit-Nummer oder, bei Mißerfolg, -1, falls alle Signal-Bits schon belegt sind. Vermerkt wird ein benutztes Signal-Bit im privaten Eintrag »tc_SigAlloc« als gesetztes Bit. »FreeSignal()« schließlich hebt die Reservierung wieder auf. Man sollte niemals versuchen, allgemein gültige Annahmen darüber zu treffen, welches Signal-Bit wozu verwendet wird. Da Exec diese vollkommen dynamisch verwaltet, müssen sie über das genannte Schutzprotokoll angefordert und auch wieder freigegeben werden.

Warten auf Signale

Ein Task tritt u.a. in den Wartezustand, indem er die Funktion »Wait()« mit einer Bit-Maske aufruft, in der alle zu erwartenden Signal-Bits gesetzt sind. Diese Funktion liefert beim Auftreten von wenigstens einem der gewünschten Signale die entsprechende Bit-Maske der tatsächlich aufgetretenen Signale zurück und löscht gleich darauf die Signale in tc_SigRecvd. Aus der Sicht eines fremden Tasks erfolgt die Signalisierung eines Ereignisses über die »Signal()«-Funktion, welche als Parameter die Adresse der Task-Struktur und die Signalmaske erwartet. Wie bereits erwähnt, verbraucht ein im Wartezustand befindlicher Task keine Prozessorzeit. Er wird der TaskReady-Liste entnommen und in die TaskWait-Liste eingegliedert. Im Eintrag tc_SigWait sind diejenigen Bits gesetzt, auf deren Eintreffen gewartet wird. tc_SigRecvd enthält die Bit-Maske der aufgetretenen Signale. Explizites Setzen oder Löschen eigener Signale erfolgt per SetSignal()-Funktion. Zwei Parameter erwartet sie: Im ersten gibt man den neuen Status der Signale in Form einer Bit-Maske an (gelöscht oder gesetzt), im zweiten ebenfalls eine Bitmaske, welche die – wenn gesetzt – betroffenen Signale kennzeichnet.

Im Zusammenhang mit der Multitaskingsteuerung ist zu erwähnen, daß ein Forbid- oder Disable-Status durch alle expliziten und impliziten Aufrufe von Funktionen, die den Task in einen Wartezustand versetzen, natürlich durchbrochen werden. Würde Exec Ausnahmebearbeitungen und Multitasking trotz Wartezustand nicht wieder zulassen, käme es zwangsläufig zu einem »Deadlock« – dem Stillstand des Systems. Warum? Der Task wartet auf ein externes Ereignis, das gar nicht mehr auftreten kann. Nach Beendigung des Wartezustands tritt der Task allerdings sofort wieder in den entsprechenden Sonderstatus ein. Ein beabsichtigter Schutzprotokoll-Effekt, für welchen Zweck auch immer, ist natürlich mit dem Eintritt in den Wartezustand hinfällig. Niemand kann vorhersehen, welche Tasks oder Ausnahmeverarbeitungen in der Zwischenzeit zum Zuge kommen können.

tc_SigExcept schließlich enthält die Maske der Signale, die bei ihrem Auftreten eine »Task-Exception«, also eine Exec-spezifische Ausnahmebehandlung (nicht identisch mit Prozessorexceptions, [1], S. 473) auslösen. Ähnliches gilt für »Traps«, die über die Einträge »tc_TrapAlloc« und »tc_TrapAble« verwaltet werden. Die vier Funktions- und Datenzeiger »tc_ExceptData«, »tc_ExceptCode«, »tc_TrapData« und »tc_TrapCode« sind ebenfalls für die Ausnahmeverarbeitungen vorgesehen.

Wichtige Task-Funktionen

Name/Offset	Aufruf und Parameter
FindTask -294	struct Task *FindTask(STRPTR) task=FindTask(name) D0 A0
SetTaskPri -300	BYTE SetTaskPri(struct Task *, LONG) oldPri=SetTaskPri(task,newpri) D0 A0 D0
Forbid -132	void Forbid(void)
Permit -138	void Permit(void)
Disable -120	void Disable(void)
Enable -126	void Enable(void)
AllocSignal -330	BYTE AllocSignal(BYTE) signalNum=AllocSignal(signalNum) D0 D0
FreeSignal -336	void FreeSignal(BYTE) FreeSignal(signalNum) D0
Wait -318	ULONG Wait(ULONG) signals=Wait(SignalSet) D0 D0
Signal -324	void Signal(struct Task *, ULONG) Signal(task,signals) A0 D0
SetSignal -306	ULONG SetSignal(ULONG,ULONG) oldSigs=SetSignal(newSigs,SigMask) D0 D0 D1
AddTask -282	APTR AddTask(struct Task *,APTR,APTR) newtask=AddTask(task,startPC,finalPC) A0 A1 A2 A3
RemTask -288	void RemTask(struct Task *) RemTask(task) A1

Speicher stapelweise

Die drei folgenden Zeiger dienen der Verwaltung des Stapelspeichers eines Tasks. Beim MC680x0 werden Stacks von oben nach unten angesprochen: »Oben« bezeichnet dabei die höchste Adresse, »Unten« die niedrigste des Speichersegments, auf welches die Zeiger »tc_SPUpper« (oben) und »tc_SPLower« (unten) verweisen. Der aktuelle Wert des Stack-Zeigers wird im Zeiger »tc_SPReg« abgelegt. Im neu initialisierten (d.h. noch unbenutzten) Stack ist diese Adresse mit der oberen Grenze identisch. Das Ablegen von Operanden auf den Stack erniedrigt den Wert des Stack-Zeigers, das Herunternehmen dagegen erhöht ihn um den Betrag der Operandenlänge.

Die Klinkenputzer

Zwei weitere Einträge in der Task-Struktur sind von besonderem Interesse: »tc_Launch« und »tc_Switch«. Diese Adressen werden immer dann angesprungen, wenn der Task den Zugriff auf den Prozessor erhält (launch = starten) oder ihn verliert (switch = umschalten). Dies geschieht allerdings nur, wenn im Eintrag »tc_Flags« die korrespondierenden Flagbits »TF_SWITCH« bzw. »TF_LAUNCH« gesetzt sind. Zu speziellen Zwecken kann es durchaus sinnvoll sein, vor und nach jedem Zugriff auf den Prozessor – unabhängig vom eigentlichen Programmlauf – nochmal in eine Routine zu springen, in der etwa Initialisierungs- und Aufräumungsarbeiten erledigt werden können. Für den normalen Gebrauch sind diese Einträge aber weitestgehend bedeutungslos und sollten auf Null gesetzt werden.

Die Liste »tc_MemEntry« dient zur Verwaltung des Speicherplatzes, den der Task anfordert. Hier können Strukturen vom Typ »MemList« verkettet werden. Die Besonderheit: Dieser Speicher wird automatisch freigegeben, wenn Exec den Task nach Beendigung wieder entfernt. Sinnvollerweise werden hier zunächst der Speicherplatz für die Task-Struktur selbst und den notwendigen Stack abgelegt.

Der letzte Eintrag, »tc_UserData«, wird vom Amiga-Betriebssystem ignoriert und kann für eigene Zwecke verwendet werden.

Nachdem nun die äußere Struktur eines Tasks klar ist, fehlt noch das Wichtigste: Wie werden Tasks gestartet und wieder entfernt? Hierzu bietet die Exec-Library die Funktion »AddTask()« an. Sie erhält neben einem Zeiger auf die Task-Struktur die Start- und Endadresse als Argumente. Als Endadresse kann ebensogut Null übergeben werden; Exec sieht für diesen Fall einen voreingestellten Ausweg am Ende der Laufzeit vor. Entfernt wird ein Task via »RemTask()«, die sich mit der Adresse der Task-Struktur als Parameter begnügt.

Im ersten Beispielprogramm wird gezeigt, wie ein Programm einen eigenen Tochter-Task ins Leben ruft und beide mit den vorgestellten Signalmechanismen kommunizieren.

Das zweite Programm ist noch ausführlicher. Auch hier startet das Programm einen Tochter-Task. Vom Hauptprogramm lassen sich nun Nachrichten per Tastatur an den Tochter-Task übergeben. Dieses Spiel kann nach einer Bestätigung seitens des Tochter-Tasks (Schalter »Reply« betätigen) fortgesetzt werden, bis eine leere Eingabezeile beide Tasks beendet. Zusätzlich besteht die Möglichkeit, vom Tochter-Task verschiedene Synchronisationsstrategien zu erproben: Einmal das zu bevorzugende »Signal-Waiting«, und – als leider nicht seltenes Negativbeispiel – das multitaskingfeindliche »Busy-Waiting«.

Der Tochter-Task zeigt in seinem Fenster ständig an, wieviel Prozent der verfügbaren Prozessorzeit tatsächlich von ihm beansprucht werden und wie oft pro Sekunde er von Exec Kontrolle über den Prozessor erlangt. Eine dazu geeignete Zeitbestimmung erfolgt über das Timer-Device in den Mantelroutinen »Switch« und »Launch«.

Wir hoffen, daß Ihnen beim Start des Programms das nervöse Hochzählen der Prozessorbelastung beim Umschalten auf »Busy-Waiting« in Erinnerung bleiben wird und als Motivation zur konsequenten Ausnutzung der Exec-Signalmechanismen dient. ■

Literaturhinweise:

[1] Commodore-Amiga, Inc.: Amiga ROM Kernel Reference Manual. Libraries. Third Edition 1992; Addison-Wesley, ISBN 0-201-56774-1

Typografie

Bullet schlägt zu

Schriften und Schnitte sind für den DTP-Profis das Nonplusultra. Nun gibt es im neuen Betriebssystem sogar eine Library, mit der man neue Schriften quasi programmieren und alte besser handhaben kann, die »bullit.library«. Wir zeigen Ihnen, was man mit ihr anfangen kann. Folgen Sie uns in die faszinierende Welt der Typografie.

von Ilse u. Rudolf Wolf

Die englischsprachige Fachliteratur nennt ein individuelles Symbol in einem Zeichensatz (Font) »Glyph«, was soviel wie geschnittener Stein bedeutet. Die ebenfalls oft verwendete Bezeichnung »Character« (Zeichen) ist nicht ganz zutreffend, denn in manchen Sprachen kann ein Glyph mehr als nur einen Teil des Alphabets darstellen. So besteht das geschriebene Chinesisch aus Bildern, die jeweils einen Begriff darstellen.

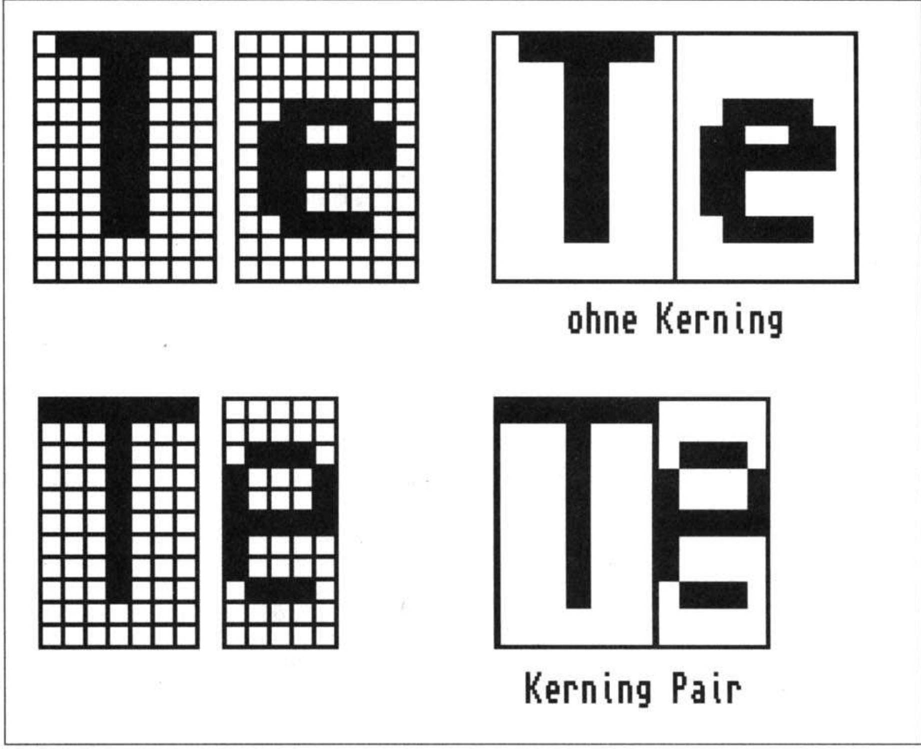
In der arabischen Schrift schreibt man keinen Text, sondern Ligaturen, die jeweils eine Zeichengruppe repräsentieren. Auch in unseren Zeichensätzen gibt es nicht nur Zeichen, sondern auch Ligaturen. Dazu kommen wir noch bei der Erläuterung des Begriffs »Kerning«. So gesehen ist es daher sinnvoll, wenn wir sagen, ein Font ist eine Sammlung von Glyphs in einem bestimmten Schriftbild (Typeface) und einer Schriftgröße. Demnach ist eine Font-Familie eine Sammlung von Fonts, die sich voneinander nur durch den Schriftstil (kursiv, fett usw.) unterscheiden.

Und Schnitt bitte

Heute treten viele Schriften in verschiedenen Varianten auf, die der deutschsprachige Typograf »Schnitte« nennt, denn jede Variation mußte neu »geschnitten« werden. Gebräuchlich sind die Abwandlungen Italic (kursiv), Bold (fett) und fett-kursiv. Beim Amiga können diese Varianten vom Betriebssystem durch algorithmische Umstellung erzeugt werden. Aus schriftgestalterischer Sicht ist ein solcher kursiver Schnitt nicht als gelungen zu betrachten, denn eine echte Kursiv-Schrift »läuft«, eine algorithmisch Geschrägte fällt dagegen um.

Mit Gutenberg gings los

Schon bevor Gutenberg die Erfindung der beweglichen Drucklettern gelang, haben unzählige Schriftzeichner Schriften kreiert. Gutenberg orientierte sich an den handgeschriebenen Büchern der damaligen Zeit. Seine Lettern waren Holzschnitte. Deren Schnitt war daher sehr wuchtig.
Erst mit dem Übergang vom Holz- zum Kupferstich wurden die Linien der Buchstaben feiner und kontrastreicher. Gleichmäßige Strichstärken bei allen Linien – Hauptlinien wie Serifen – brachte erst der Stahlstich.



»Typographie«: Man spricht von »Unterschneiden«, wenn Buchstaben wie »T« und »e« näher als üblich zusammenrücken, um häßliche Lücken zu vermeiden.

Fett ist nicht fett

Ähnliches gilt auch für die fetten Schnitte, denn es reicht nicht aus, die Linien der Buchstaben dicker zu machen. Damit schrumpfen die Buchstaben zusammen und die Lesbarkeit nimmt ab. Echte fette Schnitte beanspruchen daher auch mehr Raum.

Das Schriftbild wird auch vom Kerning beeinflußt. Setzt man nämlich Zeichen paarweise nebeneinander, ergeben sich unschöne Lücken. Um dem abzuhelfen, legt man für einen bestimmten Schnitt fest, welche Buchstabenpaare (Pair-Kerning) einen veränderten Abstand einnehmen. Z.B. lassen die Zeichen »T« und »e« im Wort Text zwischen sich eine breite Lücke, weil der Querbalken des »T« (die Laufweite bestimmend) den Abstand zum »e« bestimmt. Schiebt man das »e« ein wenig unter das »T«, verschwindet die störende Lücke. In der Typografie nennt man das »unterschneiden« (Bild). Das erfordert die gesonderte Behandlung so eines Falls. Das Pair-Kerning legt für solche Buchstabenpaare einen Korrekturwert fest. Zeichensätze dieser Art werden auch Proportional-Schriften genannt.

Im Computer werden Fonts »bitmapped« oder »outlined« gespeichert. Die Glyphs von Bitmap-Fonts werden als Bilder gesichert, während ein Outline-Font nur die Umrisse der

Glyphs und wie diese gefüllt werden sollen, beschreibt. Schauen wir uns einmal im einzelnen an, wie die einzelnen Arten im Detail aussehen:

Bitmap-Fonts:

Soll eine Schrift am Computer aufbereitet werden, ist zu berücksichtigen, daß die meisten Ausgabegeräte digital arbeiten und das Bild eines Buchstabens aus einzelnen Pixeln zusammensetzen (Rasterung). Je höher die dabei verwendete Auflösung ist, um so weniger nimmt das menschliche Auge die Umsetzung wahr.

Die Rasterung kann zu verschiedenen Zeitpunkten stattfinden. Entweder durch den Hersteller, der die Schrift in verschiedenen Schrifthöhen liefert, oder erst bei der Verwendung am Ausgabegerät (Bildschirm oder Drucker).

Bei der ersten Methode bemüht sich der Designer, Pixelversionen (Bitmaps) der Schrift zu erzeugen, die möglichst nah an den ursprünglichen Umrißlinien der Buchstaben liegen. Bei kleinen Schriftgrößen ist das aber sehr schwierig, denn wie zeichnet man Rundungen und Serifen, wenn das ganze Zeichen nur acht Pixel hoch ist? Ferner müssen die Größen der Schrift einzeln gestaltet werden. Würde man nämlich nur eine Schrifthöhe entwerfen und deren Bitmap umskalieren, würden Fehler nicht nur übernommen, sondern zusätzlich auch noch verstärkt werden.

Magischer Keks

Bis zur Workbench 1.3 verwendete der Amiga nur Bitmap-Zeichensätze. Für jede Schrifthöhe eines verfügbaren Zeichensatzes gibt es eine Bitmap-Datei, welche die Daten enthält, die benötigt werden, um die Größe dieses Zeichensatzes zu erzeugen.

Alle vom System unterstützten Zeichensätze haben auch ein sogenanntes FontContents-File (erkennbar am Suffix ».font«), aus dem ein Programm den Zeichensatz und die Schriftgröße erkennt und damit weiß, wie der Font genützt werden kann.

Das FontContents-File ist eine (definiert in <diskfont/diskfont.h>) FontContentsHeader-Struktur. Das erste Wort in dieser Struktur enthält ein sog. »Magic Cookie«, welches den Font-Typ identifiziert:

```
FCH_ID  -> 0f00 ;FontContents Bitmap-Font
TFCH_ID -> 0f02 ;TFontContents
OFCH_ID -> 0f03 ;TFontContents Outline-Font
```

Die folgende Tabelle zeigt, wie ein FontContents-File aufgebaut ist:

FontContents-File

Länge	Bedeutung
WORT	Magic Cookie
WORT	Anzahl der Schriftgrößen
256 Byte	Name des 1. Fonts
WORT	Höhe (Schriftgröße)
BYTE	Stil
BYTE	Flags (siehe Tabelle)
256 Byte	times/13 Name des 2. Fonts
....	

Outline-Fonts:

Professionelle DTP-Programme verwenden daher Schriften, die in einem Umrißformat (Outline-Format) vorliegen. Hier wird der Schnitt durch Anweisungen für Linien, Kreise und Kurven beschrieben, welche für jede Schriftgröße in das Raster des Ausgabegerätes umgesetzt werden. Die Rasterungs-Software kann in das Betriebssystem integriert sein, als separates Programm (z.B. Fontmanager) laufen oder erst im Drucker erfolgen.

Schriften im Outline-Format haben den Vorteil, daß sie auf beliebige Größen ohne Qualitätsverlust skaliert werden können. Outline-Fonts verfügen nicht über separate Dateien für jede Schriftgröße. Statt dessen konvertiert eine sog. Font-Engine über mathematische Formeln den zugrundeliegenden Zeichensatz in die gewünschte Größe. Der Zugriff auf einen Umriß-Zeichensatz nimmt wohl Rechenzeit in Anspruch, dafür aber erscheinen die Buchstaben auf dem Bildschirm genau so wie später im Ausdruck, unabhängig vom Druckertyp.

Wenn jedoch die ganze Erscheinung der Zeichen wichtiger als ihr Umriß ist, sind Bitmap-Fonts gegenüber Outline-Fonts im Vorteil. Das gilt besonders für ColorFonts, die auch am Amiga verwendet werden können (zum Beispiel mit DPaint).

Die Font-Engine »Intellifont«

Das Intellifont-Format der Agfa-Compugraphic Outline-Fonts wurde zunächst im MS-DOS-Bereich und von den Druckern der »HP LaserJet III«-Familie verwendet. Derzeit gibt es in diesem Format ungefähr 250 Fonts. Zum Amiga werden ab der Workbench 2.04 drei davon mitgeliefert.

Auf der WB 2.1 und WB 3.0 gibt es das Dienstprogramm Intellifont (auf der WB 2.04 Fountin benannt), das die Installation dieser Umriß-Zeichensätze auf dem Amiga verwaltet. Diese befinden sich, wie die standardmäßigen Bitmap-Zeichensätze im Verzeichnis FONTS:, bzw. auf der mitgelieferten Font-Diskette.

Zu jedem Outline-Font gibts nicht nur eine ».font«-Datei sondern zusätzlich eine ».otag«-Datei (».otag« steht für Outline-Tags). Der dazugehörige Outline-Font steht im Verzeichnis »_bullet_outlines« und hat das Suffix ».type.

Tabelle 1: Aufgeschlüsselter Hex-Dump CGTimes.otag

Offs.	TAG_USER+OT_Tag	Parameter	Label	Kommentar
0000:	80001001	00000105	OT_FileIdent	Länge des Files
0008:	80009002	000000BC	OT_Engine	zeigt auf Offset BC
0010:	80009003	000000C3	OT_Family	zeigt auf Offset C3
0018:	8000A005	000000CC	OT_BName	zeigt auf Offset CC
0020:	8000A006	000000D8	OT_IName	zeigt auf Offset D8
0028:	8000A007	000000E6	OT_BIName	zeigt auf Offset E6
0030:	80001010	00004C31	OT_SymbolSet	
0038:	80001011	224E2732	OT_YSizeFacor	
0040:	80002012	00000A2A	OT_SpaceWidth	
0048:	80002013	00000000	OT_IsFixed	
0050:	80001014	00000000	OT_SerifFlag	
0058:	80001015	00000080	OT_STemWeight	
0060:	80001016	00000000	OT_SlantStyle	
0068:	80001017	00000090	OT_HorizStyle	
0070:	80009020	000000B0	OT_AvailSizes	zeigt auf Offset B0
0078:	80001100	00000005	OT_SpecCount	5 Specs
0080:	80001101	00016954	OT_Spec1	
0088:	80009102	000000F8		zeigt auf Offset F8
0090:	80001103	00000420		
0098:	80001104	0000067E		
00A0:	80001105	00000002		
00A8:	00000000	00000000	TAG_DONE	
00B0:	0005			5 verfügbare Größen
00B2:	000A			10
00B4:	0014			20
00B6:	001E			30
00B8:	0028			40
00BA:	0032			50
BC:	62756C6C6574 00			bullet.
C3:	43472054 696D6573 00			CG Times.
CC:	43475469 6D657342 6F6C64 00			CGTimesBold.
D8:	43475469 6D657349 74616C69 63 00			CGTimesItalic.
E6:	4347 54696D65 73426F6C 64497461 6C6963 00			CGTimesBoldItalic.
F8:	43475469 6D65732E 74797065 00			CGTimes.type.

Funktionen der »bullet.library«

FUNKTION

ObtainInfoA -- Einen mit TFont und/oder die Schnitt-Metrik (glyph metrics) abfragen
ObtainInfo -- VarArgs-Form von ObtainInfoA

SYNOPSIS

```
error = ObtainInfoA(engineHandle,tagList)
ULONG ObtainInfoA(struct GlyphEngine *, struct TagItem *);
error = ObtainInfo(engineHandle, firstTag, ...)
ULONG ObtainInfo(struct GlyphEngine *, Tag, ...);
```

BEISPIEL

```
ULONG pointSize;
struct GlyphMap *glyph;
...
if (!ObtainInfo(EngineHandle, OT_Glyph, &glyph, TAG_DONE)) {
...
ReleaseInfo(EngineHandle, OT_Glyph, glyph, TAG_DONE);
}
```

FUNKTION

SetInfoA -- Den zu verwendenden Zeichensatz und/oder die Schnitt-Metrik (glyph metrics) festlegen
SetInfo -- VarArgs-Form von SetInfoA

SYNOPSIS

```
error = SetInfoA(engineHandle, tagList)
ULONG SetInfoA(struct GlyphEngine *, struct TagItem *);
error = SetInfo(engineHandle, firstTag, ...)
ULONG SetInfo(struct GlyphEngine *, Tag, ...);
```

BEISPIEL

```
if (!(error = SetInfo(EngineHandle, OT_PointHeight, fpoints,
    OT_GlyphCode, GC_daggerdbl, TAG_DONE)) {
error = ObtainInfo(EngineHandle, OT_Glyph, &glyph);
...
ReleaseInfo(EngineHandle, OT_Glyph, glyph);
}
```

FUNKTION

ReleaseInfoA -- Die mit ObtainInfoA erhaltenen Daten freigeben
ReleaseInfo -- VarArgs-Form von ReleaseInfoA

SYNOPSIS

```
error = ReleaseInfoA(engineHandle, tagList)
ULONG ReleaseInfoA(struct GlyphEngine *, struct TagItem *);
error = ReleaseInfo(engineHandle, firstTag, ...)
ULONG ReleaseInfo(struct GlyphEngine *, Tag, ...);
```

BEISPIEL

```
ULONG pointSize;
struct GlyphMap *glyph;
...
error = ObtainInfo(EngineHandle, OT_Glyph, &glyph, TAG_DONE);
...
ReleaseInfo(EngineHandle, OT_Glyph, glyph, TAG_DONE);
```

FUNKTION

OpenEngine -- Ein EngineHandle besorgen

SYNOPSIS

```
engineHandle = OpenEngine()
struct GlyphEngine *OpenEngine(void)
```

BEISPIEL

```
BulletBase = OpenLibrary("bullet.library", 0);
if (!BulletBase)
EndGame(ERROR_LibOpen, "bullet.library", 0);
EngineHandle = OpenEngine();
if (!EngineHandle)
EndGame(ERROR_InternalCall, "OpenEngine");
```

FUNKTION

CloseEngine -- Ein EngineHandle freigegen

SYNOPSIS

```
CloseEngine(engineHandle)
void CloseEngine(struct GlyphEngine *);
```

BEISPIEL

```
EndGame(code, arg1, arg2, arg3, arg3)
{
...
CloseEngine(EngineHandle);
...
}
```

Das Programm Intellifont erkennt zwei Outline-Formate: Amiga-Compugraphic und standardmäßige Compugraphic-Fonts, die FAIS-Dateien enthalten. Im letzteren Fall müssen die Dateien in das AmigaDOS-Format konvertiert werden, weil sie auf den Compugraphic-Disks im MS-DOS-Format enthalten sind. Erst nach der Konvertierung und Installation im Amiga stehen sie allen Anwenderprogrammen zur Verfügung.

Bevor ein Outline-Font von Intellifont bearbeitet wird, überprüft es, ob es im FONTS-Verzeichnis die Unterverzeichnisse »_bullet« und »_bullet_outlines gibt und ob es zum ».font«-File ein korrespondierendes ».otag«-File gibt. Wenn nicht, wird das Programm abgebrochen.

Die »bullet.library«:

Ab Workbench 2.0 gibt es eine neue »diskfont.library«, mit der Bitmap-Fonts skaliert werden können und ab der WB 2.1 zusätzlich die »bullet.library«, die als Font-Engine für Outline-Fonts fungiert:

- Rasterung eines Schriftbildes zu beliebigen vertikalen und horizontalen Auflösungen.
- Rotation der Buchstaben um die Basislinie um einen beliebigen Winkel.
- Schrägen (italicizing)) des Schriftbildes von -45 bis 45 Grad
- Zugriff auf Kerning-Tabellen zum Unterscheiden der Schnitte.
- Skalieren der Schnitte auf eine beliebige Schriftgröße durch algorithmische Umformung.

Die »bullet.library« kennt fünf Funktionen, mit denen die beschriebenen Möglichkeiten realisiert werden können, wie in der Tabelle »Funktionen der »bullet.library« zu sehen.

Die Übergabe der Parameter an die Funktionen der »bullet.library« erfolgt grundsätzlich mit Tags, welche die Metrik des Fonts bestimmen. Daher ist die ».font«-Datei eines Outline-Fonts nur 4 Byte lang und enthält lediglich den Font-Header, in der amerikanischen Literatur auch Magic Cookie genannt. Mehr ist nicht erforderlich, weil alle anderen Parameter im dazugehörigen ».otag«-File definiert werden. Dieses beschreibt den Schnitt, und das Schriftbild mit OT_Tags, wie sie in <diskfont/diskfonttag.h> beschrieben sind (siehe entsprechende Tabelle). Das ».otag«-File muß sich im gleichen Verzeichnis wie das ».font«-File befinden.

Die eigentliche Font-Datei mit der Endung ».type«, wird nach einer Methode gespeichert, die Keyname-Coding benannt ist. Am Beginn stehen Blocks mit Informationen die durch ein UWORD gekennzeichnet werden. Die danach folgenden Daten sind MS-DOS-orientiert und daher im Intel-Format (Low/High-Byte) gespeichert. Das ist genau umgekehrt wie das im Amiga verwendete Motorola-Format. Dazu ist zu bemerken, daß die Glyph-Daten nicht Kurven als Bogen oder Bezier-Kurven beschreiben, sondern eher eine Sammlung von Kurven-Punkten sind.

Die Anwendung der »bullet.library« erfordert gute Kenntnisse in C oder Assembler. Doch welche Fonts überhaupt zur Verfügung

stehen, kann man auch mit einem AmigaBASIC-Programm erfragen, denn hin und wieder leistet auch dieser Oldie unter den Programmiersprachen noch gute Dienste.

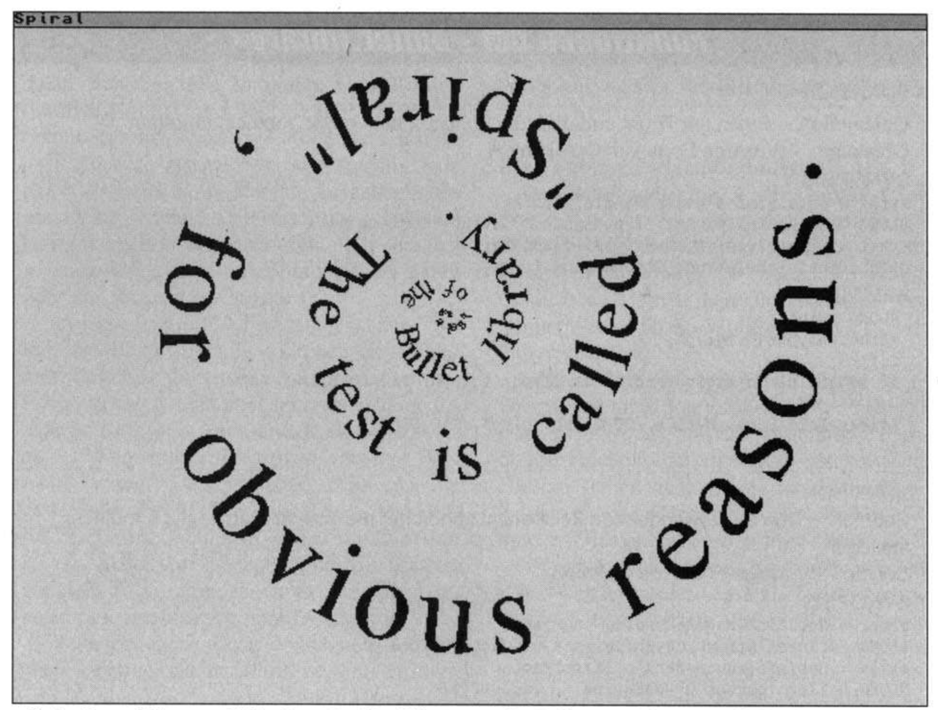
»AvailFonts.bas« (Listing unten) gibt alle verfügbaren ROM- und Disk-Fonts entweder am Bildschirm oder auf einem Drucker aus. Angezeigt werden:

- Der Name jedes Zeichensatzes und alle Schriftgrößen.
- Ob es sich um einen Bitmap- oder Outline-Font handelt.
- Der Schriftstil (siehe Tabelle 2).
- Der numerische Wert der Flags und zusätzlich dekodiert im Klartext (siehe Tabelle 3).

Auch in ARexx

Für den Fall, daß Sie AmigaBASIC nicht mehr besitzen, bieten wir Ihnen zusätzlich das ARexx-Script »FontCheck.rexx« (Listing rechts unten). Weil ARexx die AvailFonts-Funktion nicht kennt, muß man bei diesem Programm jeden Font einzeln auslesen. Als Ergebnis werden alle Disk-Fonts am Bildschirm und in die Textdatei »ram:FontCheck.txt« ausgegeben. Angezeigt werden:

- Der Name des Zeichensatzes und alle Schriftgrößen.
- Ob es sich um einen Bitmap- oder Outline-Font handelt.
- Der Schriftstil - siehe Tabelle 2.
- Bei den Bitmap-Fonts die Flags im Klartext (siehe Tabelle 3).



»Spiral_pal.lbm«. : Mit der »bullet.library« sind die verücktesten Schrifttricks programmierbar – Screenshot des Demo-Programms »Spiral.c« (auf PD-Diskette).

Auf den PD-Disketten zu diesem Sonderheft finden Sie weitere Beispiele zur Programmierung der »bullet.library«, die uns von Commodore zur Verfügung gestellt wurden (an dieser Stelle vielen Dank für die Unterstützung).

Sie finden auf der Diskette u.a. folgende Beispiele:

❒ »test.c«, Demo mit der »bullet.library«. Glyph-Metrik muß als Template (Schablone) beim Aufruf angegeben werden.

```
REM  AvailFonts.bas für AmigaBASIC 1.2

DECLARE FUNCTION AvailFonts& LIBRARY
LIBRARY "diskfont.library"
DECLARE FUNCTION AllocMem&() LIBRARY
LIBRARY "exec.library"
DEFINT a-z

DIM SHARED Fontflags$(7),textAttr&(1),style$(15)

FOR i=7 TO 0 STEP-1
 READ Fontflags$(i)
NEXT i
FOR i=0 TO 15
 READ style$(i)
NEXT i
INPUT " S)creen oder D)rucker";sd$
IF UCASE$(sd$)="S" THEN
 device$="SCRN:":sd=0
ELSE
 device$="PRT:":sd=1
END IF
CLS

'/* Speicher fuer Puffer reservieren */
bufsize&=1024
buffer&=AllocMem&(bufsize&,65537&)
mem&=buffer&
checksize&=AvailFonts&(mem&,bufsize&,7)
IF checksize&<>0 THEN
 PRINT #2, "Puffer um";checksize&;
 PRINT #2, "Bytes zu klein!"
 FreeMem& buffer&,bufsize&
 CLOSE
 END
END IF
OPEN device$ FOR OUTPUT AS #2

'/* Ueberschrift 24 32 42 */
PRINT #2,PEEKW(mem&);" Fonts ";
PRINT #2,"(Fortsezung mit Mausklick)"
PRINT #2,""
PRINT #2,"Fontname";TAB(24);"Typ";
PRINT #2,TAB(32);"Stil";TAB(42);"Flags"

'/* Fonts zeigen */
incmem = PEEKW(mem&+2)
mem&=mem&+4
WHILE incmem<>0
 FontAdr& = PEEKL(mem&)
 hoehe = PEEKW(mem&+4)
 stil  = PEEK(mem&+6)
 flags = PEEK(mem&+7)
 incmem   = PEEKW(mem&+8)
 WHILE x$<>CHR$(0)
   fontname$=fontname$+x$
   x$=CHR$(PEEK(FontAdr&))
   FontAdr&=FontAdr&+1
 WEND
 file$="Fonts:"+fontname$
 OPEN file$ FOR INPUT AS #1
  c1$=INPUT$(1,#1)
  c2$=INPUT$(1,#1):c2=ASC(c2$)
  IF c2=3 THEN c$="Outline" ELSE c$="Bitmap "
 CLOSE #1

 '/* Flags dekodieren */
 Decode flags,FontFlags$
 '/* Font-Daten zeigen */
 PRINT #2,fontname$;hoehe;TAB(24);c$;
 PRINT #2,style$(stil);" ";FontFlags$
 IF n=20 AND sd=0 THEN GOSUB warten
 mem&=mem&+10:n=n+1
 fontname$="" : x$=""
WEND
CLOSE #2
'/* Aufraeumen */
FreeMem& buffer&,bufsize&
LIBRARY CLOSE
END
warten:
CleanMouse=MOUSE(0)
WHILE MOUSE(0)=0:WEND
n=0
RETURN

SUB Decode (flags,typ$) STATIC
 typ$="":wert=flags
 FOR i=7 TO 0 STEP -1
   IF (wert-2^i)=>0 THEN
     typ$=typ$+" "+Fontflags$(i)
     wert=wert-2^i:c=1
   END IF
 NEXT i
 c=0
END SUB

DATA REMOVED,DESIGNED,PROPORTIONAL,WIDEDOT
DATA TALLDOT,REVPATH,DISKFONT,ROMFONT
DATA " PLAIN  "
DATA " UNDERLINE"
DATA " BOLD    "
DATA " UNDERLINE,BOLD"

DATA " ITALIC  "
DATA " UNDERLINE,ITALIC"
DATA " BOLD,ITALIC"
DATA " UNDERLINE,BOLD,ITALIC"
DATA " EXTENDED"

DATA " UNDERLINE,EXTENDED"
DATA " UNDERLINE,EXTENDED"
DATA " UNDERLINE,BOLD,EXTENDED"
DATA " ITALIC,EXTENDED"
DATA " UNDERLINE,ITALIC,EXTENDED"
DATA " BOLD,ITALIC,EXTENDED"
DATA " UNDERLINE,BOLD,ITALIC,EXTENDED"
```


»AvailFonts.bas« : Programm, um alle verfügbaren Zeichensätze zu ermitteln

Tabelle 2. Definition tf_Style-Byte

Wert	Label	Wirkung
0	FS,NORMAL	normal=kein Stilattribut gesetzt
1	FS,UNDERLINED	unterstrichen
2	FS,BOLD	fett (tf_BoldSmear=1)
4	FS,ITALIC	kursiv
8	FS,EXTENDED	erweitert (z.B."Topaz 9")
64	FS,COLORFONT	
128	FS,TAGGED	

Tabelle 3. Definition tf_Flags-Byte

Wert	Label	Bedeutung
1	ROMFONT	ROM-Zeichensatz
2	DISKFONT	Disk-Zeichensatz
4	REVPATH	von rechts nach links schreiben
8	TALLDOT	Font für HiRes/Non-Interlaced
16	WIDEDOT	Font für LoRes/Interlaced
32	PROPOTIONAL	Zeichenbreite variiert
64	DESIGNED	siehe Anmerkung
128	REMOVED	Font nicht aktiv

Anmerkung: Ist dieses Bit gesetzt, ist der Font entworfen (designed). Wenn nicht, kann der Font (ab »diskfont.library« Version 36 und höher) auch aus einem im ROM oder auf Diskette bereits existierenden Font skaliert worden sein.

❐ »strikefont.c«, zeigt den ganzen Zeichensatz eines OutlineFonts (Template für den Aufruf im Quellcode).

❐ »spiral.c«, zeigt weitere Möglichkeiten mit der »bullet.library«. Ergebnis siehe Screenshot »Spiral_pal.lbm«. Wahlweise auch Aufruf mit Templates.

❐ »spiral_pal«, das Original von Commodores Entwickler-Diskette enthält einen Bug im Window-Titel und die Fensterhöhe ist mit 640x400 zu klein. Das Programm wurde daher von unserem Autor Rudolf Wolf auf PAL-Höhe (640 x480) gepatcht und der Fenster-Titel korrigiert.

❐ »spacing.c«, skaliert »CGTimes.font« ab Schriftgröße 5,6,7,8... Punkte.

❐ »sofdemo.c«; eine Demo, die zeigt, wie man mit skalierbaren Outline-Fonts umgeht.

❐ »prtface.c«, dieses »typeface print utility« schließlich zeigt die Programmierung von Druckerausgaben.

Wie erwähnt, finden Sie die Demos auf den PD-Disketten zum Heft (siehe Seite 114) ■

```
/* ============= FontCheck.rexx ============= */
if(~show('l','rexxsupport.library'))
 then call addlib('rexxsupport.library',0,-30,0)
call FontTyp()
options prompt " Font-Verzeichnis? > "
pull dirname
if dirname = "" then dirname = 'FONTS:'
/* Font-Verzeichnis speichern */
fonts = showdir('FONTS:','FILES',' ')
num = words(fonts) /* Anzahl der Fonts */
/* Font-Analysen in ein File speichern */
say ' Fonts werden analysiert...'
open(sende,'ram:Fontcheck.txt','w')
do slot=1 to num
 fontname = word(fonts,slot)
 if right(fontname,4)='font'
   then pfad = 'FONTS:' || fontname
 say pfad
 /* Fontheader checken */
 open(check,pfad,'r')
  MagicCookie = c2x(readch(check,2))
  select
   /* Bitmap-font */
   when MagicCookie = '0F00' then do
    anzahl = c2d(readch(check,2))
    hoch =""
    do n=1 to anzahl
     dummy = readch(check,257)
     y = c2d(readch(check,1))
     hoch = hoch y
     stil = c2d(readch(check,1))
     flags = readch(check,1)
     klartext = ""
     do j=0 to 7 /* Flags dekodieren */
       b = bittst(flags,j)
       if b=1 then klartext = klartext typtxt.j
       end
    end
   writeln(sende,">" fontname ' FCH_ID =  0F00 -> Bitmap Font' hoch)
   writeln(sende,"  Stil: " stltxt.stil "Flags:" c2d(flags) "->" klartext)
   writeln(sende,"")
   end
   when MagicCookie = '0F02' then do
     writeln(sende,">" fontname ' TFCH_ID = 0F02 -> TFontContents')
     writeln(sende,"")
   end
   /* Outline-Font */
   when MagicCookie = '0F03' then do
     otagname = left(pfad,length(pfad)-5) || '.otag'
     open(tags,otagname,'r')
      do while ~EOF(tags)
        x = c2x(readch(tags,8))
        if x = '00000000' then leave
        end
      anzahl= c2d(readch(tags,2))
      hoch =""
      do n=1 to anzahl
        y = c2d(readch(tags,2))
        hoch = hoch y
        end
     call close(tags)
     writeln(sende,">" fontname 'OFCH_ID = 0F03 -> Outline Font' hoch)
     writeln(sende,"")
   end
   otherwise NOP
  end
  call close(check)
end
call close(sende)
/* Ergebnis am Bildschirm mit MORE ausgeben */
address COMMAND
 'echo "*ec"'
 'more ram:Fontcheck.txt'
 'echo "*ec"'
options prompt ' Fontanalyse-File löschen? -j/n >'
pull auswahl
if auswahl ='J' then do
  'delete ram:Fontcheck.txt'
  end
exit
FontTyp:
 typtxt.7 = "REMOVED"
 typtxt.6 = "DESIGNED"
 typtxt.5 = "PROPORTIONAL"
 typtxt.4 = "WIDEDOT"
 typtxt.3 = "TALLDOT"
 typtxt.2 = "REVPATH"
 typtxt.1 = "DISKFONT"
 typtxt.0 = "ROMFONT"
 stltxt.0 = "PLAIN"
 stltxt.1 = "UNDERLINE"
 stltxt.2 = "BOLD"
 stltxt.4 = "ITALIC"
return
```


»FontCheck.rexx«: Mit ARexx kann man natürlich auch mit Fonts jonglieren

ARexx-Tips

Königliche Hilfe

Schwerpunktthemen dieser Ausgabe von »Faszination Programmieren« sind der Amiga 1200, das neue Betrioebssystem und ARexx, die neue Programmiersprache für den Amiga. Im folgenden zeigen wir Ihnen, was man mit ARexx alles anfangen kann und wir starten gleich mit ein paar praktischen Tips.

Klar, das Interesse an ARexx ist groß. Mittlerweile liefert Commodore die neue Sprache mit jedem Amiga aus. Das gute alte BASIC soll ersetzt werden. Hier nun ein paar Kniffe im Umgang mit ARexx.

Falls sie selbst ein paar gute Tricks auf Lager haben, schicken Sie uns doch einfach Ihre Vorschläge auf Diskette. Im nächsten »Faszination Programmieren« wir ARexx sicher auch wieder eine große Rolle spielen.

SUPER_Rename in ARexx

Wollten Sie beispielsweise schon einmal alle Dateien namens »Mon#?.s« in »HyperMon#?.asm« in einem Verzeichnis umbenennen? Mit dem ARexx-Programm »Ren.rexx« ist es möglich. Die Benutzung ist im Programm als Kommentar enthalten. Wenn man in die »Shell-Startup« die Zeile

```
alias ren rx ren.rexx []
```

einfügt, braucht man nur noch »ren alt neu« einzugeben. Das Programm ändert die Namen aller Dateien im aktuellen Verzeichnis

Thies Wellpott

ARexx-Kill per Workbench

ARexx läßt sich – wie bekannt – per Doppelklick auf das RexxMast-Icon in der System-Schublade starten. Durch einen einfachen Trick kann man das Ganze – was weniger bekannt sein dürfte – auch wieder durch einen Doppelklick auf ein Icon deaktivieren. Speichern Sie hierzu die folgende Datei (ASCII) unter dem Namen »RexxKill«:

```
DF0:Rexxc/RXC        ; geben Sie statt DF0:
                     ; den Namen Ihrer
                     ; Workbench-Diskette ein
avail flush >nil:    ; diese Befehlsfolge
                     ; löscht nicht mehr
                     ; benötigte Librarys
                     ; aus dem Speicher
```

Nun Starten Sie den Icon-Editor (zu finden auf der Extras-Diskette, die jedem Amiga beiliegt), und laden das RexxMast-Icon. Auf Wunsch kann es verändert werden; übermalen Sie z.B. die Krone schwarz. Speichern Sie das Ergebnis als »Projekt«-Icon unter dem Namen »RexxKill.info« im selben Verzeichnis, in dem die »RexxKill«-Datei steht. Geben Sie als Default-Tool »C:IconX« an und das Icon ist einsatzbereit.

Oberbuchner Christian

```
/* rename command
 * von Thies Wellpott
 * Usage: rx ren [olddir/]oldname [newdir/][newname]
 * oldname und newname können Jokerzeichen (#?, ?, etc.) beinhalten
 * Wenn newname KEINE Jokerzeichen enthält, wird das DOS-Rename benutzt.
 * Enthält newname aber Jokerzeichen so gilt folgendes:
 * ren [olddir/][oldprefix]#?[oldsuffix] [newdir/][newprefix]#?[newsuffix]
 */
signal on BREAK_C
parse arg oldname newname .
oldjoker = check_joker(oldname)
newjoker = check_joker(newname)
if newjoker = 0 then    /* newname enthält keine Jokerzeichen */
do                               /* also DOS-Rename benutzen */
   address command 'Rename "'oldname'" "'newname'"'
   exit 0
end
if (oldjoker ~= 1) | (newjoker ~= 1) then
do
   say "Wrong number of pattern chars!"
   exit 20
end
olddir = ""
pos = get_dirpos(oldname)
if pos ~= 0 then
do
   olddir = left(oldname, pos)
   oldname = right(oldname, length(oldname)-pos)
end
newdir = ""
pos = get_dirpos(newname)
if pos ~= 0 then
do
   newdir = left(newname, pos)
   newname = right(newname, length(newname)-pos)
end
parse var oldname oldprefix "#?" oldsuffix
parse var newname newprefix "#?" newsuffix
address command 'List >t:xyz_ren.tmp QUICK NOHEAD
        LFORMAT="%S"' olddir || oldname
if open(fhd, "t:xyz_ren.tmp" ,"R") = 0 then
do
   say "Can`t open my tmp-file!"
   exit 20
end
len_op = length(oldprefix)
len_os = length(oldsuffix)
fname = readln(fhd)
do while ~eof(fhd)
   pos = lastpos(oldsuffix, fname)
   newname = newprefix || substr(fname, len_op+1,
             length(fname)-, len_op-len_os) || newsuffix
   say 'Renaming "'olddir || fname'" as "'newdir || newname'"'
   address command 'Rename "'olddir || fname'" "'newdir || newname'"'
   fname = readln(fhd)
end
call close(fhd)
address command "Delete >NIL: t:xyz_ren.tmp"
exit 0
BREAK_C:
   say "*** break"
   exit 5
procedure check_joker:
   arg name
   pos = 0
   anz = 0
   do until pos = 0
      pos = index(name, "#?", pos+1)
      anz = anz + 1
   end
   return anz-1
procedure get_dirpos:
   arg name
   pos = lastpos("/", name)
   if pos = 0 then
      pos = index(name, ":")
   return pos
```


»Ren.rex«: Ein ARexx-Script, um Dateien umzubennen, das aber auch ganze Verzeichnisse verarbeitet

Externe Bibliotheken

ARexxxxxx wächst

An Stelle von AmigaBASIC werden die neuen Amigas mit der Interpretersprache ARexx ausgeliefert. Was leistet diese Sprache? Was kann man mit ihr anfangen? Hier ein paar Beispiele, die zeigen, daß ARexx Ihnen viel Arbeit mit dem Amiga abnimmt.

von Ilse u. Rudolf Wolf

Heute muß jedes Anwenderprogramm einen ARexx-Port besitzen. Hat es keinen, gilt es fast als veraltet. Die Praxis zeigt jedoch, daß höchstens zehn Prozent der Anwender davon Gebrauch machen...

Und als eigenständige Programmiersprache wird ARexx kaum verwendet. Dabei ist der Befehlssatz mit externen Funktionsbibliotheken fast unbegrenzt erweiterbar.

Die meisten Anwender und Programmierer wissen gar nicht, was alles in ARexx steckt, deshalb wollen wir uns in dieser Ausgabe mal richtig mit den Möglichkeiten beschäftigen:

Fast unbegrenzter Befehlsumfang

Mit externen Bibliotheken wird ARexx nicht nur als Batch-Sprache zur automatisierten Steuerung des Amiga oder Makro-Sprache zur Steuerung eines Anwenderprogramms sondern auch als universelle Programmiersprache einsetzbar. Solche externe Funktionsbibliotheken findet man auf PD-Disketten:

❒ Die universellste Bibliothek dürfte die APIG (A Programmers Intuition & Graphics Library) von Fish-Disk 634 sein (s. »Faszination Programmieren Nr.1, Seite 57), denn mit ihr ist der Zugriff auf 288 Funktionen aus den Amiga-OS2-Bibliotheken (ASL, exec, GadTools, graphics, intuition, layers und utility) möglich. Zusätzlich enthält die Bibliothek über 600 Konstanten und Flags, wie sie in den Include-Files definiert sind. Kein Wunder, daß die »apig.library« 61220 Byte lang ist und das Manual rund 50 Seiten umfaßt.

Mit der APIG ist ein flexibles und funktionelles Programmieren ähnlich wie in C möglich, denn in den meisten Fällen konvertiert die »apig.library« die ARexx-Stringparameter in ein Format, welches die Amiga-Bibliotheksfunktionen erwarten. Teilweise werden auch Funktionen zu Makros zusammengefaßt, die einen kompakten Programmcode ermöglichen. Folgende Strukturen werden unterstützt:

Menu, MenuItem, SubItem, Requesters, Boolean, String- und Proportional-Gadgets, Borders, IntuiText, 16-Bit-Arrays, Layers und IFF (via »iff.library« von Christian Weber).

Allen Programmierern, die mit Intuition/Graphics-Funktionen und Strukturen nicht vertraut sind, empfiehlt der Programmierer Ronnie E. Kelly daher dringend das Studium der Includes & Autodocs.

Eine der Stärken der APIG sind die integrierten Funktionen der »GadTools.library« und damit ist der Zugriff auch von ARexx aus möglich. Diese Bibliothek dient dazu, das Programmieren von Gadgets, Menüs und Intuition-Ereignissen zu vereinfachen. Mußten bisher viele Datenstrukturen vom Programmierer erstellt werden, erfolgt das ab jetzt mit wenigen Zeilen. Dazu zeigen wir im Artikel »Gut geschaltet mit ARexx« das Beispiel »GadTools.rexx« mit den Gadget-Typen: CYCLE, BOOLEAN, RADIO BUTTONS, LISTVIEW, STRING, TEXT und PROPGADGET. Alle diese Gadgets erscheinen in einer Bevelbox, die mit »DRAWBEVELBOX()« erzeugt wird.

❒ Die mathematischen Funktionen des ARexx kann man mit der »rexxmathlib.library« von der Fish 227 ergänzen, die alle trigonometrischen Funktionen, sowie die e^x-, x^y- und die n!-Funktion liefert. Anscheinend ist diese aber nicht immer zu APIG kompatibel, denn manchmal kommt es zu Abstürzen infolge von Adreßfehlern.

❒ Eine schon ältere Bibliothek ist die »rex_intui.library« von Jeff Glatt (Fish 463). Sie kennt zwar nur Funktionen des Amiga-OS 1.3, enthält dafür jedoch einige Funktionen, welche weniger Programmieraufwand erfordern als mit der »apig.library«. Die IFF-ILBM-Funktionen der »rx_itui.library« sind sogar um einiges leistungsfähiger, weil sie auf die »ILBM.library« zugreifen und daher alle Auflösungen des Amiga beherrschen (auch HAM!).

Nachteilig ist allerdings, daß mit der »rx_intui.library« programmierte Custom-Screens (weil OS1.3) und deren Windows im alten 2-D-Look erscheinen. Abhilfe schafft hier »TagScreens« (siehe auch folgende Seite, Tabelle »TagScreens1.6«).

Bilder laden per ARexx-Script

Die »rx_intui.library« enthält nur rund 40 Funktionen. Trotzdem ist sie sehr leistungsfähig, weil fast alle dieser Funktionen Makros sind. Dazu ein Beispiel mit dem der Darstellungsmodus auf JAM2, die Zeichenfarbe auf 2 und die Hintergrundfarbe auf 0 eingestellt wird:

Mit der »apig.library«:

```
call setapen(windowrastport,2)
call setbpen(windowrastport,0)
call setdrmd(winrastport,1)
```

Und mit der »rx_intui.library«:

```
call SetDraw(window,2,0,1)
```

Ein Vergleich der Befehlsnamen zeigt, daß die der APIG und der RexxIntui nicht kollidieren. Trotzdem können Funktionen nicht problemlos in einem ARexx-Script kombiniert werden. Eine der Ursachen ist, daß APIG-Zeiger Zeichenketten sind, die von ARexx als Hex-Strings interpretiert werden, während die RexxIntui für Zeiger numerische Werte liefert. Welche Bibliothek besser geeignet ist, hängt daher von der zu programmierenden Anwendung ab.

In einigen Fällen sind Kombinationen möglich. Das zeigt das ARexx-Script »IFF-ILBM.rexx« auf der übernächsten Seite. Dort wird ein ASL-Filerequester mit der APIG programmiert, während der IFF-ILBM-Viewer mit der RexxIntui erzeugt wird. Es kommt aber zu

Quellennachweis für die Bibliotheken

Disk	Name	Funktion
227	RexxMathLib	ARexx um trigonometrische Funktionen erweitern.
348	ColorReqester	»color.library« (Doc-File auf Fish 257) -> kann von der »rx_intui.library« aus aufgerufen werden.
463	FileIO	»requester.library« -> wird von den Requestern der »rx_intui.library« benötigt.
463	ILBM	»ilbm.library« -> wird von den IFF-Funktionen der »rx_intui.library« benötigt.
463	RexxIntuition	Zugriff auf diverse OS1.3-Intuition-Funktionen ermöglichen.
634	APIG	Ermöglicht den Zugriff auf Exec-, Graphics-, Intuition-, Layers-, Asl-,Utility- und Gadtools-Funktionen (OS2). Die »iff.library« wird mitgeliefert.
705	MFR/Goodies	Tagscreens -> Patcht mit OS1.3 programmierte Screens.

keinen Kollisionen, weil die APIG nach dem Gebrauch des Filerequesters aus der Bibliotheksliste entfernt wird und erst danach die »RexxIntui.library« aktiviert wird.

Wie erwähnt greifen einige Funktionen dieser Bibliotheken auf andere externe Libs zu. Um für alle Fälle gerüstet zu sein, sollten daher die »iff.library«, »ilbm.library«, »color.library« und »requester.library« zur Verfügung stehen. Die Tabelle auf der vorigen Seite zeigt eine Liste. Ein voller »Fish«-Korb, nicht wahr?

Diesen Korb selbst füllen zu müssen, wollen wir Ihnen ersparen. Auf den Begleitdisketten (Public Domain) zu diesem Heft finden Sie daher alle beschriebenen Bibliotheken.

Bibliotheken sind Public

Die Verzeichnisse APIG, RexxIntuition und RexxMathLib enthalten (weil Kopien der Originale von den Fish-Disks 634,463 und 227) auch die Dokumentation und Programmbeispiele der Autoren in gepackter Form. Die Entpacker dazu gibt es im Verzeichnis C. Dort ist auch das erwähnte »TagScreens« gespeichert.

❐ Der Vollständigkeit halber sei noch die »RexxArpLib« von W.G.J. Langeveld (Fish 227) genannt. Mit ihr ist der Zugriff auf die noch immer unter OS1.3 gerne verwendete »ARP-Library« möglich.

Leider enthält die RexxArpLib einige Bugs, die nicht mehr beseitigt wurden, weil der Autor die Bibliothek nicht mehr weiterentwickelt hat.

Startvorbereitungen

Bevor APIG-Funktionen aufgerufen werden, müssen die vom Programm benötigten externen Funktionsbibliotheken (APIG, RexxIntui, RexxSupport usw.) in die ARexx-internen Bibliothekenliste aufgenommen werden.

Ronnie Kelly macht das in seinen APIG-Programmbeispielen mit

```
call addlib('apig.library',0,-30,0)
```

oder

```
x = addlib('apig.library',0,-30,0)
```

und setzt dabei stillschweigend voraus, daß die »rexxsupport.library« bereits geladen wurde. Ist das nicht der Fall, so brechen viele seiner Beispiele mit einer Fehlermeldung ab, weil sie Funktionen aus dieser Bibliothek enthalten.

Wir empfehlen daher in den APIG-Beispielen des Autors den ADDLIB-Aufruf durch die folgende Sequenz zu ersetzen:

```
if(~show('l','rexxsupport.library'))
 then call addlib('rexxsupport.library',
                  0,-30,0)
if(~show('l','apig.library'))
 then call addlib('apig.library',0,-30,0)
```

Hier wird abgefragt, ob die Bibliotheken in die Liste eingebunden sind. Wenn nein, werden sie in den Speicher geladen und in die Liste aufgenommen.

Jeff Glatt setzt in allen seinen Beispielen voraus, daß die »rx_intui.library« bereits geladen wurde und seine Scripts enthalten daher keinen Aufruf. Wenn Sie daher die Beispiele

TagScreens 1.6

Dieses Dienstprogramm patcht _LVOOpenScreen und _LVOOpenScreenTagList. Dadurch erscheinen auch alle unter OS1.3 programmierten Screens im 3D-Look des OS2. TagScreens ist daher besonders für alle AmigaBasic-Anwender interessant, die diese Sprache am Amiga 500+ oder Amiga 600 verwenden.

Zu finden ist TagScreens1.6 (mit Source und Doc-File) auf der Fish 705 im Verzeichnis MFR/Goodies und auf den PD-Disketten zu dieser Ausgabe.

Hinweis: Aufgerufen wird TagScreens aus der Shell (im Normalfall ohne Parameter). Ein nochmaliger Aufruf macht den Patch rückgängig.

Achtung: Es wird Kickstart V37.175 oder höher benötigt!

```
/* Externe Bibliotheken aktivieren */
 if(~show('l','rexxsupport.library'))
  then call addlib('rexxsupport.library',0,-30,0)
 if(~show('l','apig.library'))
  then call addlib('apig.library',0,-30,0)
 if(~show('l','rexxmathlib.library'))
  then addlib('rexxmathlib.library',0,-30,0)

/* Intuition-Konstanten global aktivieren */
call SET_APIG_GLOBALS()
portname = "apig1_port"
p = openport(portname)

/* <> OPENWINDOW(portname,left,top,wid,hgt,dpen,bpen,IDCMP,
       flags,title,scr,console,bitmap,chkmark,gadlist */

wx = 640;wy = 256
title = " Ellipsen"
winidcmp = CLOSEWINDOW
flags = WINDOWCLOSE+WINDOWDRAG+WINDOWSIZING+,
        WINDOWDEPTH+GIMMEZEROZERO+ACTIVATE
w1 = openwindow(portname,0,0,wx,wy,2,0,winidcmp,flags,title,
     null(),0,0,0,0)
rpw1 = getwindowrastport(w1)

/* Kreis mit DRAWELLIPSE(rp,cx,cy,a,b) */
call setapen(rpw1,3)
call DRAWELLIPSE(rpw1,320,120,200,100)

/* Ellipse mit  DRAWCIRCLE(rp,cx,cy,r) */
call setapen(rpw1,2)
call DRAWCIRCLE(rpw1,320,120,120)
call setapen(rpw1,1)

/* Unterprogramm Elipse */
call setapen(rpw1,1)
call Ellipse(320,120,200,80,0,360,0,2)
call setapen(rpw1,2)
call Ellipse(320,120,200,80,0,180,45,2)
call setapen(rpw1,3)
call Ellipse(320,120,200,80,180,360,45,2)
call setapen(rpw1,1)
call Ellipse(320,120,200,80,0,360,135,2)
exitme = 0
do while exitme = 0
     x = waitpkt(portname)
     do forever
        msg = getpkt(portname)
        if msg = '0000 0000'x then leave
        msgclass  = getarg(msg,0)
        x = reply(msg,0)
        select
           when msgclass = CLOSEWINDOW then exitme = 1
           otherwise nop
        end
     end
end
call CLOSEWINDOW(w1)
exit

Ellipse:
 arg xm,ym,a,b,aw,ew,rw,asp
 bm=3.141593/180;rw=rw*bm
 aw=aw*bm ;ew=ew*bm
 stp=0.1  ;ew=ew+stp
 xw=COS(rw);yw=SIN(rw)
 ym=ym*asp
  x=a*COS(aw);y=b*SIN(aw)
  x1=x*xw-y*yw+xm;y1=(x*yw+y*xw+ym)/asp
  x1=trunc(x1);y1=trunc(y1)
 do i=aw to ew by stp
  x=a*COS(i);y=b*SIN(i)
  x2=x*xw-y*yw+xm;y2=(x*yw+y*xw+ym)/asp
  x2=trunc(x2);y2=trunc(y2)
   call move(rpw1,x1,y1)
   call draw(rpw1,x2,y2)
  x1=x2;y1=y2
 end
return
```


»Ellipsen.rexx«: Wir erweitern APIG um eine Funktion zum Zeichnen von Ellipsen

des Autors ausprobieren wollen, müssen Sie den ADDLIB-Aufruf ergänzen:

```
if(~show('l','rx_intui.library'))
  then addlib('rx_intui.library',0,-30,0)
```

Es gehört zu einem sauberen Programmierstil, externe Bibliotheken am Programmende mit der REMLIB-Funktion wieder aus der Liste zu entfernen. Damit sind die Libraries zwar aus der ARexx-Liste entfernt, bleiben aber im RAM. Bei Speicherknappheit kann man belegten Speicher mit dieser Zeile freigeben:

```
address COMMAND 'avail >NIL: flush'
```

In Ronnie Kelly's Beispielen gibt´s noch einen Bug: Er verwendet den AmigaDOS-Befehl WAIT. ARexx kennt ihn nicht und interpretiert z.B. »wait 5 secs« nicht als Befehl. In den Beispielen müssen daher WAIT-Zeilen durch einen Zugriff auf AmigaDOS ergänzt werden:

```
wait 5 secs
```

durch

```
address COMMAND 'wait 5 secs'
```

Wenn die »rexxsupport.library« geladen ist, geht es kürzer und schneller mit:

```
call delay(250)
```

Weil DELAY als Argument die Ticks/sek verlangt, muß für 5 Sekunden der Wert 250 eingesetzt werden.

Libraries für ARexx im Einsatz

Wir haben für Sie einige Beispiele programmiert. Damit Sie diese nicht abtippen müssen, sind sie im Verzeichnis »Scripts« der Begleitdiskette (s. Seite 114) enthalten. Alle Scripts sind mit Icons versehen und können auch von der Workbench aus gestartet werden. Um Ihnen langatmige Erklärungen zu ersparen, sind alle Listings reichlich kommentiert.

Ellipsen.rexx (erstes Listing)

Ergänzt die APIG um eine Ellipse-Funktion, die auch Rotation und das Zeichnen von Bögen mit wählbarem Anfangs- und Endwinkel ermöglicht. Die dazu erforderlichen Parameter stehen als Kommentar im Listing.

IFFILBM.rexx (Listing 2)

Dieses Beispiel zeigt, wie man die APIG und die RexxIntui miteinander kombiniert und liefert einen IFF-ILBM-Viewer für alle Formate (auch HAM!). Er wird mit der »RexxIntui« + »ILBM.library« erzeugt. Für die Bildauswahl dient der ASL-Filerequester (APIG). Die Farben des Bilds können mit einem Color-Requester verändert werden (Aufruf mit rechter Maustaste) und unter einem neuen Namen wieder gespeichert werden. Beendet wird das Programm mit der linken Maustaste.

Diese Kombination wird verwendet, weil APIG zwar den ASL-Filerequester kennt, jedoch die »iff.library« von Christian Weber verwendet, die HAM nicht beherrscht. RexxIntui dagegen nutzt die »ILBM.library«, die alle Standard-Bildformate verarbeitet.

Menu_A.rexx und Menu_I.rexx (Disk)

Zeigen jeweils eine Variante, wie man Menüs mit der APIG (Menu_A.rexx) und der RexxIntui (Menu_I.rexx) programmiert.

AslFont.rexx (Listing 3, nächste Seite)

Beweist, daß man auch in ARexx mit Hilfe der APIG den Zeichensatz wechseln kann. Das Script ist modular aufgebaut, so daß man es eigenen Erfordernissen anpassen kann. ■

Literaturnachweis:
Commodore-Amiga,Inc.,West Chester, Pennsylvania:
Eric Giguére - AMIGA Programmer's Guide to ARexx
Commodore: Handbuch zur Systemsoftware 2.0
Commodore: Includes & Autodocs V39.108


```
/* ============ IFFILBM.rexx ============
 IFF-ILBM-Viewer mit der rx_intui.library
   für: Hi-Res, Lo-Res, Interlace, HAM.
       Zusätzlich erforderlich sind:
 ilbm.library, apig.library, color.library
===================================== */
/* APIG einbinden */
if(~show('l','apig.library'))
  then call addlib('apig.library',0,-30,0)
/* Intuition-Konstanten setzen */
call set_apig_globals()
/* Tagliste für ASL-Filerequester anlegen */
asltags = makeasltaglist()
/* ASL-Filerequester aufrufen */
pic = ASLREQUEST(freq,asltags)
if pic = '0000 0000'x then do
 say "Requester CANCELED!"
end
call FREEASLREQUEST(freq)
call remlib('apig.library')
if pic = '0000 0000'x then exit

/* Bild laden (rx_intui) */
 if(~show('l','rx_intui.library'))
   then addlib('rx_intui.library',0,-30,0)
 win1=IffLoad(0,1+2,pic)
 if win1 = 0 then say 'Kein IFF-ILBM File!'

 /* IDCMP abfragen */
 scrn=PEEK(win1,46,2)
 call ModIDCMP(win1,,,65536)
 call workloop()

/* Aufräumen */
call EndWindow(win1)
call EndScreen(scrn)
call remlib('rx_intui.library')
address COMMAND 'avail >NIL: flush'
exit
workloop:
 ans=0
 do forever
  msg=WaitMsg(win1)
  PARSE var msg class part1 part2 part3
  if class = 3 then do
   /* Mausklick beendet */
   if part1 = 0 then leave
   /* Menü-Taste -> Color-Requester aufrufen */
   if part1 = 2 then do
    call Color(scrn)
    ans=MsgThree(win1,' Bild speichern? ',' OK oder NO',)
   end
  end
  if ans=1 then leave
 end
if ans=1 then do
  win2=GetWindow('Eingabe',scrn,20,20,280,40,,,)
  err = SetDraw(win2,2,,0)
  gadg2=AddGadg(win2,4,,10,12,250,12,1,1,2,40)
  err=SetDraw(win2,3,,1)
  spec=WaitMsg(win2)
  pfad = GInfo(win2,1)
  err=EndWindow(win2)
  err=IFFsave(win1,pfad)
  call workloop()
end
return

makeasltaglist:
    freq = ALLOCASLREQUEST(ASL_FILEREQUEST,'0000 0000'x)
    asltags = makepointer(freq,0,200,MEMF_CLEAR)
    hailstring = makepointer(freq,0,80,MEMF_CLEAR)
    call export(hailstring,"Select A File(s)... or CANCEL")
    initpattern = makepointer(freq,0,80,MEMF_CLEAR)
    call export(initpattern,"~(#?.info)")
    initdir = makepointer(freq,0,80,MEMF_CLEAR)
    call export(initdir,"ram:")
    call SETTAGSLOT(asltags,0,ASL_HAIL,'p',hailstring)
    call SETTAGSLOT(asltags,1,ASL_LEFTEDGE,'n',20)
    call SETTAGSLOT(asltags,2,ASL_TOPEDGE,'n',20)
    call SETTAGSLOT(asltags,3,ASL_WIDTH,'n',380)
    call SETTAGSLOT(asltags,4,ASL_HEIGHT,'n',190)
    call SETTAGSLOT(asltags,5,ASL_PATTERN,'p',initpattern)
    call SETTAGSLOT(asltags,6,ASL_DIR,'p',initdir)
    aslfuncflags = FILF_NEWIDCMP + FILF_PATGAD
    call SETTAGSLOT(asltags,7,ASL_FUNCFLAGS,'n',aslfuncflags)
    call SETTAGSLOT(asltags,8,ASL_WINDOW,'p','0000 0000'x)
    call SETTAGSLOT(asltags,9,TAG_DONE,'n',0)
return asltags          © 1993 M&T
```

»IFFILBM.rexx«: Ein Programm, um Bilder anzuschauen – für alle Bildschirmmodi!!

```
/* ============= AslFont.rexx ============= */
/*  Screen, Window, Fontrequester mit Tags  */

/* Externe Bibliotheken einbinden */
if(~show('l','rexxsupport.library'))
  then call addlib('rexxsupport.library',0,-30,0)
if(~show('l','apig.library'))
  then call addlib('apig.library',0,-30,0)

/* Intuition-Konstanten setzen */
call set_apig_globals()

/* Messageport öffnen */
portname = "msgport"
p = openport(portname)
WaitForPort portname

/* Screen-Tags definieren */

scrtaglist = makescrtaglist()
/* Screen öffnen */
scr  = openscreentaglist(null(),scrtaglist)

/* Window-Tags definieren */
wintaglist = makewintaglist()

/* Window öffnen */
win = openwindowtaglist(portname,null(),wintaglist)
winrastport = getwindowrastport(win)

/* Zeiger auf den aktuellen Font */
sysfont = getvalue(winrastport,52,4,'p')

/* Requester-Tags definieren */
call makeasltaglist()

/* Fontrequester aufrufen u. Auswahl im Fenster zeigen */
do forever
 req = aslrequest(freq,asltags,freq)
 if req = null() then leave
 fontname = getvalue(freq,8,4,'s')
 size     = getvalue(freq,12,2,'n')
 fg       = getvalue(freq,16,1,'n')
 bg       = getvalue(freq,17,1,'n')
 drmd     = getvalue(freq,18,1,'n')

 /* Selektierten Font zeigen */
 call pitext(winrastport,10,60,fontname size,fg,bg,drmd,req)

 /* ürsprünglichen Font wieder setzen */
 call setfont(winrastport,sysfont)
 cal pitext(winrastport,10,190,"Bitte warten..",2,1,
            JAM2,sysfont)
 call delay(200) /* 4 Sekunden warten */

 /* Bildschirm löschen */
 call clearscreen(winrastport,0,0)
end
say " CANCEL angeklickt!"

/* Aufräumen */
call closewindow(win)
call closescreen(scr)
call freeaslrequest(freq)
call freetagitems(wintaglist)
call freetagitems(scrtaglist)
call freevec(mytitle)
/*
call remlib('rexxsupport.library')
call remlib('apig.library')
address COMMAND 'avail >NIL: flush'
*/
exit
/* ---------------------------------------------------- */

makescrtaglist:
   scrtagl = allocatetagitems(18)
   sapens  = makepointer(scrtagl,0,24,MEMF_PUBLIC)
   call setvalue(sapens,0,2,'n',-1)
   scrpname = makepointer(scrtagl,0,80,MEMF_PUBLIC)
   call export(scrpname," Screen mit 16 Farben")
   call settagslot(scrtagl,0,SA_Title,'p',scrpname)
   call settagslot(scrtagl,1,SA_Left,'n',0)
   call settagslot(scrtagl,2,SA_Top,'n',0)
   call settagslot(scrtagl,3,SA_Width,'n',640)
   call settagslot(scrtagl,4,SA_Height,'n',256)
   call settagslot(scrtagl,5,SA_DISPLAYID,'n',HIRES)
   call settagslot(scrtagl,6,SA_PENS,'p',sapens)
   call settagslot(scrtagl,7,SA_Depth,'n',4)
   call settagslot(scrtagl,8,SA_Type,'n',CUSTOMSCREEN)
   call settagslot(scrtagl,9,TAG_DONE,'n',0)
return scrtagl

makewintaglist:
   winidcmp = IDCMP_CLOSEWINDOW
   winflags = WFLG_CLOSEGADGET + WFLG_DRAGBAR +
              WFLG_SIZEGADGET,
              + WFLG_DEPTHGADGET + WFLG_GIMMEZEROZERO
   wintitle = " Font Outputwindow"

   mytitle = allocvec(length(wintitle)+1,MEMF_PUBLIC)
   call export(mytitle,wintitle)
   wintagl = allocatetagitems(20)
   call settagslot(wintagl,0,WA_LEFT,'n',0)
   call settagslot(wintagl,1,WA_TOP,'n',12)
   call settagslot(wintagl,2,WA_WIDTH,'n',640)
   call settagslot(wintagl,3,WA_HEIGHT,'n',220)
   call settagslot(wintagl,4,WA_DETAILPEN,'n',1)
   call settagslot(wintagl,5,WA_BLOCKPEN,'n',0)
   call settagslot(wintagl,6,WA_IDCMP,'n',winidcmp)
   call settagslot(wintagl,7,WA_FLAGS,'n',winflags)
   call settagslot(wintagl,8,WA_TITLE,'p',mytitle)
   call settagslot(wintagl,9,WA_CUSTOMSCREEN,'p',scr)
   call settagslot(wintagl,10,WA_MINWIDTH,'n',40)
   call settagslot(wintagl,11,WA_MAXWIDTH,'n',640)
   call settagslot(wintagl,12,WA_MINHEIGHT,'n',80)
   call settagslot(wintagl,13,WA_MAXHEIGHT,'n',255)
   call settagslot(wintagl,14,WA_SIZEGADGET,'n',1)
   call settagslot(wintagl,15,WA_DRAGBAR,'n',1)
   call settagslot(wintagl,16,WA_CLOSEGADGET,'n',1)
   call settagslot(wintagl,17,WA_ACTIVATE,'n',1)
   call settagslot(wintagl,18,WA_GIMMEZEROZERO,'n',1)
   call settagslot(wintagl,19,TAG_DONE,'n',0)
return wintagl

makeasltaglist:
 /* 8 bytes per slot  */
 freq = allocaslrequest(ASL_FONTREQUEST,null())
 asltags = makepointer(freq,0,72,MEMF_CLEAR)

  /* Begrüßungstext */
 hailstring = makepointer(freq,0,80,MEMF_CLEAR)
 call export(hailstring,"Select A Font... or CANCEL")
 call settagslot(asltags,0,ASL_HAIL,'p',hailstring)

  /* Zeiger auf das Fenster */
 call settagslot(asltags,1,ASL_WINDOW,'p',win)
 call settagslot(asltags,2,ASL_LEFTEDGE,'n',100)
 call settagslot(asltags,3,ASL_TOPEDGE,'n',24)
 call settagslot(asltags,4,ASL_FRONTPEN,'n',1)
 call settagslot(asltags,5,ASL_BACKPEN,'n',0)

  /* Schrifthöhe max. 48 Pkte erlaubt */
 call settagslot(asltags,6,ASL_MAXHEIGHT,'n',48)

  /* Funcflags definieren */
 aslfuncflags = FONF_FRONTCOLOR       /* fg color palette */
 aslfuncflags = aslfuncflags + FONF_BACKCOLOR /* bg color
palette*/
 aslfuncflags = aslfuncflags + FONF_STYLES /* style buttons */
 aslfuncflags = aslfuncflags + FONF_DRAWMODE
                              /* drawmode gadget */

 /* Wenn ASL_WINDOW definiert ist, dann muß
    auch FONF_NEWIDCMP gesetzt werden (sonst GURU!) */
 aslfuncflags = aslfuncflags + FONF_NEWIDCMP
 call settagslot(asltags,7,ASL_FUNCFLAGS,'n',aslfuncflags)
 call settagslot(asltags,8,TAG_DONE,'n',0)
return asltags  © 1993 M&T
```

»AslFont.rexx«: Fontwechsel mit ARexx programmiert; wie Sie sehen, mit ARexx geht alles...

ARexx und die »GadTools.library«

Gut geschaltet mit ARexx

Die GadTools-Library (ab OS2.0) dient dazu, das Programmieren von Gadgets, Menüs und Intuition-Ereignissen zu vereinfachen. Mußten hierfür bisher viele Datenstrukturen vom Programmierer erstellt werden, kann das jetzt mit wenigen Aufrufen der Funktionen der »GadTools.library« erfolgen. Die wichtigsten Funktionen dieser Bibliothek wurden von Ronnie Kelly in seine APIG-Library übernommen und damit ist der Zugriff auch von ARexx aus möglich.

von Ilse u. Rudolf Wolf

Unser Beispiel »GadTools.rexx« (unten) zeigt, wie man in ARexx Gadgets programmiert. Es verwendet folgende Gadget-Typen: Cycle, Boolean, Radio Buttons, Listview, String und Text. Alle Gadgets erscheinen in einer Bevelbox, die mit »DRAWBEVELBOX()« erzeugt wird.

Als Draufgabe enthält das Beispiel noch ein Proportional-Gadget in dem der Schieber horizontal und vertikal verschoben werden kann.

Die GadTools-Funktionen brauchen einen Speicher, in dem Daten aufbereitet werden können. Das geschieht mit der APIG-Funktion »CREATECONTEXT()«. Ferner werden auch einige Zeiger benötigt (siehe Listing).

Alle Gadgets eines Windows sind miteinander verkettet. Der Aufbau einer solchen Gadget-Kette ist aus dem reichlich kommentierten ARexx-Script ersichtlich.

Den Beginn der Kette bildet eine Gadget-Struktur, die mit der Funktion »MAKENEWGADGET()« initialisiert wird. Erst der Aufruf von »CREATEGADGET()« besorgt ein Gadget des angegebenen Typs und hängt es an ein bestehendes Gadget an.

Bevor allerdings mit »CREATEGADGET()« weitere Gadgets erzeugt und eingebunden werden können, muß die vorher mit »MAKENEWGADGET()« initialisierte Gadget-Struktur mit »SETNEWGADGET()« modifiziert werden.

Ronnie E. Kelly, der Programmierer der APIG-Library, verweist in seiner Dokumentation ständig auf das RKM (Rom Kernel Reference Manual). Damit ist der Anwender gezwungen, sich dort Detailinformationen über Flags und TagItems zu holen.

Letzteres wollen wir Ihnen ersparen und liefern in der Tabelle auf der nächsten Seite ergänzende Informationen zu allen wichtigen Funktionen, die es Ihnen ermöglichen, das Script nach eigenen Wünschen zu modifizieren bzw. einzelne Gadgets in eigenen ARexx-Scripts zu verwenden. Damit es nicht zu unübersichtlich wird, werden Funktionen, deren Parameter aus dem Listing ersichtlich sind, hier nicht erwähnt.

Damit Sie 250 Zeilen »GadTools.rexx« nicht abtippen müssen, ist das Programm auf der Begleitdiskette gespeichert. Das Script ist mit einem Icon versehen und kann daher auch von der Workbench aus gestartet werden.

Ferner finden Sie auf der Begleitdiskette im Verzeichnis LIBS alle Bibliotheken. Die Verzeichnisse APIG, RexxIntuition und RexxMathLib enthalten (weil Kopien der Originale von den Fish-Disks 634, 463 und 227) auch die Dokumentation und Programmbeispiele der Autoren in gepackter Form. Die Entpacker dazu gibt es im Verzeichnis C. ■

```
/* ========= GadTool.rexx ======== */
/* Beispiele mit GadTool-Funktionen */

 /* Externe Bibliotheken aktivieren */
 if(~show('l','rexxsupport.library'))
  then call addlib('rexxsupport.library',0,-30,0)
 if(~show('l','apig.library'))
  then call addlib('apig.library',0,-30,0)
/* Intuition-Konstanten global aktivieren */
call SET_APIG_GLOBALS()

/* ExecList für LISTVIEW-Gadget anlegen */
gadexeclist = makelist()

/* Labels für CYCLE- u. MX- Gadgets erzeugen */
cyLabels = makeCYlbls()
mxLabels = makeMXlbls()

/* Gadgets definieren */
z = DefineGads()

/* Fenster öffnen  */
portname = "apiggad_port"
p = openport(portname)

WaitForPort portname
wintitle = " GadTools - Beispiele / KLICK ME !"
winidcmp = CLOSEWINDOW+GADGETUP+GADGETDOWN+MOUSEMOVE
winflags = WINDOWCLOSE+WINDOWDRAG+WINDOWSIZING+WINDOWDEPTH+GIMMEZEROZERO,
           +ACTIVATE
w1  = OPENWINDOW(portname,0,0,640,256,0,1,winidcmp,winflags,wintitle,
                 ,scr,0,null(),null(),conxgad)
rpw1 = GETWINDOWRASTPORT(w1)

/* BevelBox als Umrandung aller Gadgets zeichnen */
call DRAWBEVELBOX(rpw1,10,4,570,180,scrvinfo,TAG_DONE,0)
/* BevelBox als Umrandung des PropGadgets zeichnen */
call DRAWBEVELBOX(rpw1,134,200,312,38,scrvinfo,TAG_DONE,0)

/* Alle GadTools-Gadgetes neu zeichnen */
call GT_REFRESHWINDOW(w1,null())

/* PropGadget einbinden */
z = addglist(w1,gprop,-1,-1,0)
z = refreshgadgets(gprop,w1,0)

/* Überschrift MX Gadget */
z = pitext(rpw1,230,10,"   MX_KIND",1,0,JAM2,0)
z = pitext(rpw1,230,20,"(RADIO BUTTONS)",1,0,JAM2,0)

/* Überschrift BOOLEAN Gadget */
z = pitext(rpw1,32,56,"BOOLEAN_KIND",1,0,JAM2,0)

/* Auf CLOSE-Gadget oder CANCEL warten */
exitme = 0; apen=1
do while exitme = 0

 x = waitpkt(portname)
 do forever
    msg = getpkt(portname)
    if msg = '0000 0000'x then leave
    msgclass  = getarg(msg,0)
    msgcode = getarg(msg,1)
    msgid = getarg(msg,9)
    msgY = getarg(msg,4)
    x = reply(msg,0)
    if msgclass = CLOSEWINDOW | msgid=14 then do
      if msgid=14
       then gtxt=" CANCEL"
       else gtxt=" Das CLOSE-Gadget"
     say  gtxt "wurde angeklickt!"
      exitme=1
    end
```

»GadTools.rexx«: Ein ARexx-Programm, mit dem Sie Gadget ausprobieren können (Anfang)

```
    /* Status des angeklickten Gadgets ausgeben */
    a = auswertung()
  end
end

/* Aufräumen */
call CLOSEWINDOW(w1)

call FREEGADGETS(conxgad)
call FREETHIS(newgadx)
call FREEVEC(glistpointer)
call FREETHIS(cyLabels)
call FREETHIS(mxLabels)
call FREE_EXEC_LIST(gadexeclist,,1)

call FREEVISUALINFO(scrvinfo)
call UNLOCKPUBSCREEN(null(),scr)
call remlib('apig.library')
call remlib('rexxsupport.library')
address COMMAND 'avail >NIL: flush'

exit

DefineGads:
/*  Von den GadTools-Funktionen benötigte Zeiger initialisieren */
 scr          = LOCKPUBSCREEN("Workbench")  /* Window am WB-Screen */
 scrvinfo     = GETVISUALINFO(scr)          /* für MAKENEWGADGET() */
 scrfont      = GETVALUE(scr,40,4,'p')      /* Zeiger auf TextAttr */
 glistpointer = ALLOCVEC(4,MEMF_CLEAR)
 conxgad      = CREATECONTEXT(glistpointer) /* Zeiger auf Context */
 previousgadget = conxgad
 myid = 10                                  /* GadgetID */

/* CYCLE-Gadget - GadgetID=10 (Beginn der Kette) */
newgadx = MAKENEWGADGET(scrvinfo,scrfont,30,32,190,16,"CYCLE_KIND",
                        ,PLACETEXT_ABOVE,myid,null())
previousgadget = CREATEGADGET(CYCLE_KIND,previousgadget,newgadx,
                              ,GTCY_LABELS,cyLabels,
                              ,TAG_DONE,0)

/*  MUTUAL EXCLUDE Gadget (Radio Butttons) - GadgetID =11 */
call SETNEWGADGET(newgadx,scrvinfo,scrfont,240,32,145,42,"",
                  ,PLACETEXT_RIGHT,myid+1,null())
previousgadget = CREATEGADGET(MX_KIND,previousgadget,newgadx,
                              ,GTMX_LABELS,mxLabels,
                              ,TAG_DONE,0)

/* LISTVIEW-Gadget GadgetID=12 */
call SETNEWGADGET(newgadx,scrvinfo,scrfont,370,22,185,120,"LISTVIEW_KIND",
                  ,PLACETEXT_ABOVE,myid+2,null())
previousgadget = CREATEGADGET(LISTVIEW_KIND,previousgadget,newgadx,
                              ,GTLV_LABELS,gadexeclist,
                              ,LAYOUTA_SPACING,1,
                              ,TAG_DONE,0)

/* STRING-Gadget GadgetID=13 */
call SETNEWGADGET(newgadx,scrvinfo,scrfont,30,94,290,12,"STRING_KIND",
                  ,PLACETEXT_ABOVE,myid+3,null())
previousgadget = CREATEGADGET(STRING_KIND,previousgadget,newgadx,
                              ,GTST_MAXCHARS,50,TAG_DONE,0)

/* BOOLEAN-Gadget GadgetID=14  */
call SETNEWGADGET(newgadx,scrvinfo,scrfont,30,68,70,12,"CANCEL",
                  ,PLACETEXT_IN,myid+4,null())
previousgadget = CREATEGADGET(BUTTON_KIND,
                 previousgadget,newgadx,TAG_DONE,0)

/* SLIDER-Gadget GadgetID=0 */
stext="SILDER_KIND 0 - 100"
call SETNEWGADGET(newgadx,scrvinfo,scrfont,30,120,290,12,stext,
                  ,PLACETEXT_ABOVE,myid+10,null())
previousgadget = CREATEGADGET(SLIDER_KIND,previousgadget,newgadx,
                              ,GTSL_MAX,100,TAG_DONE,0)

/* SCROLLER-Gadget GadgetID=0 */
sctxt="SCROLLER_KIND 0 - 20"
call SETNEWGADGET(newgadx,scrvinfo,scrfont,30,150,520,12,sctxt,
                  ,PLACETEXT_ABOVE,myid+20,null())
previousgadget = CREATEGADGET(SCROLLER_KIND,previousgadget,newgadx,
                              ,GTSC_ARROWS,16,
                              ,GTSC_TOP,10,
                              ,GTSC_VISIBLE,1,
                              ,GTSC_TOTAL,21,
                              ,TAG_DONE,0)

/* TEXT Gadget zur Anzeige der Rückagbewerte    */
xt=" TEXT_KIND -> Status des angeklickten Gadgets"

call SETNEWGADGET(newgadx,scrvinfo,scrfont,30,166,520,12,"",
                  ,PLACETEXT_ABOVE,myid+40,null())
previousgadget = CREATEGADGET(TEXT_KIND,previousgadget,newgadx,
                              ,GTTX_TEXT,xt,
                              ,GTTX_BORDER,1,
                              ,TAG_DONE,0)

  /* --- Proportional-Gadget mit Funktionen der Intuition-Library --- */
  /* PROP GADGET mit  AutoKnob in Bevelbox */
  maxnumX=200
  maxnumY=10
  gprop  = makepropgadget(newgadx,140,204,300,30,GADGHNONE,
         GADGIMMEDIATE+RELVERIFY,
         +FOLLOWMOUSE,-1,FREEVERT+FREEHORIZ+AUTOKNOB+16,,,myid+30,0,0)
  txt1="PropGadget x -> 0-200 y -> 0-10"
  call makeitext(w1,txt1,AUTOLEFTEDGE,-14,1,0,JAM2,0,gprop)
return 1

/* Texte für das Cycle Gadget */
makeCYlbls:
      ltxt.1="Farbe 1 (SCHWARZ)"
      ltxt.2="Farbe 2 (WEISS)"
      ltxt.3="Farbe 3 (HELLBLAU)"

      /* Array 4 Pointers 4Bytes * 4 = 16  */
      cylbl = MAKEPOINTER(0,0,16,MEMF_CLEAR)

      /* Nur bis 3, weil das Letzte Null sein muß (Listenabschluß) */
      do x = 1 to 3
         lbuf = MAKEPOINTER(cylbl,0,length(ltxt.x)+2,MEMF_CLEAR)
         call export(lbuf,ltxt.x)
         call SETVALUE(cylbl,(x-1)*4,4,'p',lbuf)
      end
return cylbl

/* Texte für die Radio Buttons */
makeMXlbls:

      /* Array 6 Pointers 4Bytes * 4 = 24  */
      maLbls = MAKEPOINTER(0,0,44,MEMF_CLEAR)

      /* Nur 5, weil das Letzte Null sein muß (Listenabschluß) */
      do x = 1 to 5
         ltxt = "Schalter" x
         lbuf = MAKEPOINTER(maLbls,0,length(txt)+2,MEMF_CLEAR)
         call export(lbuf,ltxt)
         call SETVALUE(maLbls,(x-1)*4,4,'p',lbuf)
      end
return maLbls

/* Texte für Listview Gadget */
makelist:

   /* List Struktur (immer  ALLOCMEM benützen!)  */
   listnode = ALLOCMEM(18,MEMF_CLEAR)
   call NEWLIST(listnode)
   do x = 1 to 20
      txt = "Das ist Zeile " x
      ptr_to_node_added = ADD_LIST_NODE(listnode,txt)
   end
return listnode

auswertung:
 txt=""
 item = msgcode +1
 select

   when msgid = 0 & msgY <210 then do
     if msgY <130
       then txt = "Position Slider Gadget ->" msgcode
       else txt ="Position Scroller Gadget ->" msgcode
     end

   when msgid=40 | msgid=0 then do
     x=maxnumX * trunc(horizpot(gprop)*100 /65535)/100
     y=maxnumY * trunc(vertpot(gprop)*100 /65535)/100
     txt="PropGadget: X=" x "Y=" y
     end
```

»GadTools.rexx«: Ein ARexx-Programm, mit dem Sie Gadget ausprobieren können (Fortsetzung)

```
   when msgid = 10 then do
     apen= msgcode +1
     cyTxt = GETVALUE(cyLabels,msgcode*4,4,'s')
     txt="Cycle Gadget ->" cyTxt
     end

   when msgid = 11 then do
     rbTxt = GETVALUE(mxLabels,msgcode*4,4,'s')
     txt="Radio Buttons ->" rbTxt
     end

   when msgid = 12 then do
     nodeptr=gadexeclist

     do for item
      nodeptr = getvalue(nodeptr,0,4,'p')
     end
     nodename = getvalue(nodeptr,10,4,'s')
     txt = "Listview Gadget ->" nodename

     end

   when msgid = 13 then do
     item= GETSTRGAD(w1,13,0)
     txt = "String Gadget ->" item
     end
   otherwise NOP
 end

 call pitext(rpw1,38,168,Copies(" ",62),1,0,JAM2,0)
 call pitext(rpw1,38,168,txt,apen,0,JAM2,0)

return 1                    © 1993 M&T
```

»GadTools.rexx«: Ein ARexx-Programm, mit dem Sie Gadget ausprobieren können (Ende). Alle Skripts und erforderlichen Libraries finden Sie auch auf den PD-Disketten zum Heft (siehe Seite 114).

Tags für CREATEGADGET()

Aufruf und Parameter	Funktion
CYCLE_KIND	
GTCY_Active	; Nummer des aktiven Labels (default=0)
GTCY_Labels	; Mit Null abgeschlossenes Label-Feld
LISTVIEW_KIND	
GTLV_Labels	; Zeiger auf die Liste der Texte
GTLV_ScrollWidth	; Breite des Scroll-Balkens (default=16)
GTLV_Top	; Nummer des ersten sichtbaren Eintrags
LAYOUTA_SPACING	; Bestimmt den Zeilenabstand
MX_KIND	
GTMX_Active	; Aktive Nummer im MX-Gadget (default=0)
GTMX_Labels	; Mit Null abgeschlossenes Label-Feld
GTMX_Spacing	; statt LAYOUTA_SPACING bei MX_KIND
SCROLLER_KIND	
GTSC_Arrows	; Pfeilgröße
GTSC_Top	; Anfangsposition (default=0)
GTSC_Total	; Anzahl der Einheiten (default=0)
GTSC_Visible	; Auf einmal sichtbare Einheiten
SLIDER_KIND	
GTSL_Level	; Anfangsposition (default=0)
GTSL_Max	; Maximalwert des Sliders
GTSL_Min	; Minimalwert des Sliders
STRING_KIND	
GTST_MaxChars	; Maximal zulässige Zeichen
GTST_String	; Inhalt (default= leer)
TEXT_KIND	
GTTX_BackPen	; Hintergrundfarbe
GTTX_Border	; 1 = Rahmen zeichnen
GTTX_FrontPen	; Textfarbe
GTTX_Text	; Inhalt (default = leer)

Wichtigste Funktionen der »GadTools.library«

Aufruf und Parameter	Funktion
MAKENEWGADGET(vinfo,font,left,top,width,height,text,flags,id,usrdata)	Erzeugt eine NewGadget-Struktur für GadTool-Gadgets. Die Freigabe muß mit der FREETHIS()-Funktion erfolgen. Rückgabewert: Zeiger auf die Struktur
SETNEWGADGET(newgad,vinfo,font,left,top,width,height,text,flags,id,usrdata)	Modifiziert die Werte in einer vorher initialisierten NewGadget-Struktur. Rückgabewert: 1 wenn erfolgreich, sonst 0 Erklärung der Parameter für MAKENEWGADGET und SETNEWGADGET: flags – Flags für Text-Position (PLACETEXT_LEFT/PLACETEXT_RIGHT/PLACETEXT_IN/PLACETEXT_ABOVE) font – Zeiger auf den Zeichensatz, font = GETVALUE(screen,40,4,'p') height – Höhe des Gadgets id – GadgetID left – x-Koordinate der linken oberen Ecke newgad – Zeiger auf die mit MAKENEWGADGET() initialisierte NewGadget-Struktur text – string, Text für das Gadget-Label top – y-Koordinate der linken oberen Ecke usrdata – Zeiger (z.B. auf ein Image) vinfo – Der von der GETVISUALINFO()-Funktion gelieferte Zeiger width – Breite des Gadgets
CREATEGADGET(kind,previous,newgad,tag1,tag1data,...)	Der eigentliche Befehl, das Gadget zu zeigen kind – Gadget-Typ newgad – Zeiger auf die mit MAKENEWGADGET() initialisierte NewGadget-Struktur
Gadget-Typen (Tags siehe Tabelle oben rechts):	
BOOLEAN_KIND	Liefert ein Boolean-Gadget, das lediglich durch Anklicken aktiviert wird. Es kann zwei Zustände annehmen, aktiv oder inaktiv.
LISTVIEW_KIND	Liefert ein Listen-Gadget mit einem Scroll-Balken, das Einträge in einer Liste verwaltet und die durch Anklicken der Zeile ausgewählt werden.
MX_KIND	Liefert sog. Radio-Buttons, das sind kleine quadratische Schalter, die im aktiviertem Zustand ausgefüllt sind. Sie können mit anderen Radio-Buttons so verknüpft werden, daß ein Button abhängig vom Zustand anderer Buttons aktiviert oder deaktiviert wird. Die Art der Verknüpfung nennt man »mutually exclusive«
CYCLE_KIND	Liefert ein Cycle-Gadget mit dem eine Liste verschiedener Optionen durchgeklickt werden kann. Beispiele dafür finden sich auf der Workbench 2.xx und 3.0.
SCROLLER_KIND	Erzeugt einen Scroll-Balken.
SLIDER_KIND	Erzeugt einen Schieberegler.
STRING_KIND	Diesen Typ gab es bereits in älteren Betriebssystem-Versionen, doch jetzt erscheint er am Bildschirm im neuen 3-D-Look.
TEXT_KIND	Liefert ein Gadget, das nur Text enthält und sonst keine weiteren Funktionen erfüllt.

ARexx-Port zum Nulltarif

Endlich kommt :

Text-Editoren für den Amiga gibt es (besonders im PD-Bereich) wie Sand am Meer. Einer befindet sich auch auf Ihrer Workbench-Diskette. Doch wußten Sie, daß der Editor freidefinierbare Pull-Down-Menüs und einen leistungsfähigen ARexx-Port besitzt? Wir verraten Ihnen, wie Sie ihn bedienen und um zusätzliche Funktionen erweitern – zu einem idealen Werkzeug für Programmierer.

von Heinzjörg Rabe

Unser hier vorgestellter »Ed 2.00«, im folgenden kurz »ED« genannt, befindet sich unter dem Namen »Ed« im Directory »C« Ihrer Workbench-Diskette. Die aktuelle Version ist »Ed 2.00, V37.11, 13.5.91«, die ab Workbench-Version 2.04 (37.67) ausgeliefert wird, und nur unter OS 2.0 und höher läuft.

ED ist ein bildschirmorientierter ASCII-Text-Editor. Zur Anzeige verwendet ED ein Console-Window (»RAW:«) auf dem Workbench-Screen. Vorgängerversionen von ED befinden sich seit den Anfängen des Amiga im Lieferumfang (auf der Workbench-Diskette), erfreuten sich aber keiner großen Beliebtheit.

Fundamentale Änderungen am AmigaDOS, die beim Sprung von Kickstart 1.3 nach OS 2.0 erfolgt sind, machten eine Überarbeitung von »ED« erforderlich. Dabei wurden – glücklicherweise – gleich noch einige Verbesserungen vorgenommen. Hier die wesentlichsten Verbesserungen gegenüber der Version 1.14:

- voll OS 2.0 kompatibel (kein BCPL mehr!);
- von der Workbench aus startbar,
- zusätzliche Aufrufparameter bzw. Tool-Types erlauben u.a. Definition von Windowgröße, Tabulatoren, Margins;
- Dateiauswahl über File-Requester der ASL-Library;
- Cursorpositionierung mit der Maus;
- freidefinierbare Roll-Down-Menüs;
- Unterstützung von Macro-Files;
- freidefinierbare Funktionstasten;
- ARexx-Fernbedienung aller Funktionen,
- Start von ARexx-Scripts über File-Requester.

Der Start

Da ED die File-Auswahl über den File-Requester der »asl.library« ermöglicht, muß sich die Datei »asl.library« im Directory »LIBS:« befinden. ED kann von CLI/Shell, als auch von der Workbench gestartet werden. Allerdings befindet sich im Lieferumfang kein Icon; das entsprechende File »Ed.info« fehlt.

Wenn Sie ED von der Workbench (Doppelklick) starten möchten, brauchen Sie ein Icon vom Typ »Tool« erzeugen. Natürlich können Sie für Ihre Text-Icons vom Typ »Project« wählen. Tragen Sie dort »c:Ed« als »Default Tool« ein. Beim Doppelklick auf so ein Project-Icon lädt ED nach dem Start den Text.

Seit OS 2.0 durchsucht die Workbench, um das zu einem Icon gehörende Programmfile zu finden, übrigens auch den Suchpfad (PATH-Befehl und mit ASSIGN zugewiesenes Directory »C:«). Es genügt also, wenn Sie das selbstgebastelte Tool-Icon »Ed.info« z.B. ins Utilities-Directory Ihrer Boot-Diskette kopieren und das eigentliche Programm »Ed« im »C:«-Directory belassen. Für den Start vom CLI/Shell lautet das Template (die Schablone):

```
Ed "FROM/A,SIZE/N,WITH/K,WINDOW/K,TABS/N,
   WIDTH=COLS/N,HEIGHT=ROWS/N"
```

Hierzu einige Beispiele:

```
Ed SYS:s/startup-sequence
Ed Riesentext SIZE 100000 TABS 8
Ed Ram:Text WINDOW "RAW:0/0/640/256/
  Ed 2.00/CLOSE/NOSIZE"
```

Bei Workbench-Benutzung können mit »Information« aus dem »Icons«-Menü der Workbench entsprechende »Tool Types« im Icon »Ed.info« bzw. den Project-Icons Ihrer Texte eingetragen werden. Deren Bedeutung ist:

```
FROM   - Name des zu editierenden Textes
SIZE   - Textpuffergröße (mind. 12000)
WITH   - Macro-Script für Konfiguration
WINDOW - Windowdefinition für DOS-Funktion
         »Open()«
TABS   - Tabulator-Schrittweite (normal 3)
WIDTH  - Rechter Rand (Right Margin)
HEIGHT - genutzte Zeilenanzahl
```

Gibt man keine Parameter oder Tool-Types an, gelten folgende Defaultwerte:

```
SIZE   60000
WITH   "S:Ed-Startup"
WINDOW "RAW:0/0/639/199/Ed 2.00/CLOSE"
TABS   3
WIDTH  77
HEIGHT 23
```

Durch explizite Angabe einer Window-Definition können Sie das ED-Window auch in voller Bildschirmgröße oder sogar auf einem Public-Screen öffnen.

Nachdem ED sein Window geöffnet hat, versucht der Amiga, die Datei »S:Ed-Startup« zu laden und als Macro-Script auszuführen (s.u.). Dieses File enthält in der Regel Anweisungen für globale Voreinstellungen von ED, z.B. Definition der Roll-Down-Menüs, Funktionstasten-Belegungen, etc.

Wird die Datei nicht gefunden, werden Default-Einstellungen vorgenommen. Interessanterweise sind die Default-Roll-Down-Menüs wesentlich umfangreicher, als die in »S:Ed-Startup«, weshalb wir dem erfahrenen ED-Benutzer raten, dafür zu sorgen, daß ED diese Datei nicht findet (z.B. umbenennen), oder ein eigenes »Super«-WITH-File zu erstellen (s.u.).

Wurde beim Programmstart mit dem Parameter »WITH« ein weiteres Macro-Scriptfile angegeben, wird es nach dem automatischen Laden des ersten Textes beim Start zusätzlich ausgeführt. Man kann so entweder weitere Voreinstellungen vornehmen, oder ED gewissermaßen selbständig Texte bearbeiten lassen.

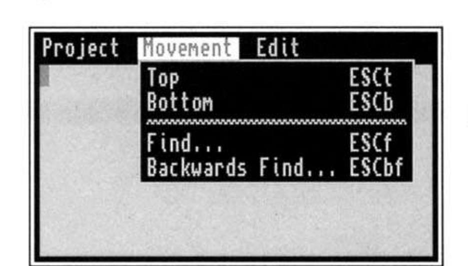
»Neuer ED«: Wußten Sie, daß der einfach stungsstarke Befehle und Menüs etc. verfü

ED bearbeitet Texte selbständig

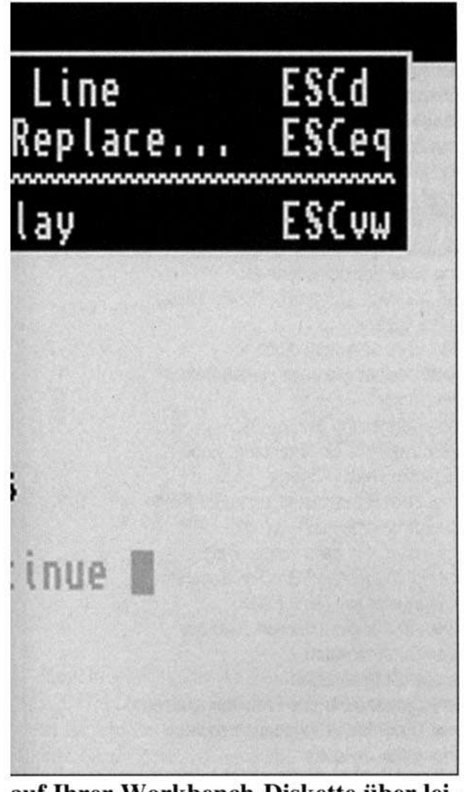
Der Ladevorgang

Wie dem »/A« beim Template zu entnehmen ist, muß beim Start vom CLI/Shell immer ein Textfilename angegeben werden, das Schlüsselwort »FROM« kann jedoch auch entfallen. Am einfachsten ist es, wenn man beim Start von ED den Namen eines nicht existierenden Files in der RAM-Disk angibt (»RAM:Dummy«) und das eigentliche Textfile dann bequem durch den File-Requester auswählt.

Während des Ladens zeigt ED die Füllung des Textpuffers durch Klammeraffen »@« an, deren Anzahl proportional zur Ausnutzung des



ch ED zu Ehren



auf Ihrer Workbench-Diskette über lei- ARexx ist alles möglich.

(durch »SIZE« wählbaren) Puffers ist. (Rechter Windowrand = Puffer voll.)

Jedes Textfile wird bereits während des Ladens einem Filterprozeß unterworfen. Hierbei verändert sich folgendes:

- Alle Zeichen mit ASCII-Codes in den Bereichen $01-$08, $0B, $0E-$1A, $1C-$1F und $80-$9F führen zum Abbruch des Ladevorgangs mit der Meldung »File contains binary«.
- Die Zeichen $00, $0C (FF = FormFeed), $0D (CR = Carriage Return) und $1B (ESC = Escape) werden in $0A (LF = LineFeed) umgewandelt!
- Tabulator-Sprünge (TAB = $09) werden immer erweitert, d.h. je nach dem angegebenen Parameter »TABS« bis zum nächsten Tabulator-Stop mit Leerzeichen aufgefüllt.
- Ferner werden alle Zeilen spätestens nach dem 255. Zeichen zwangsbeendet (truncated = abgeschnitten).

Die Veränderungen werden ggf. nach dem Laden in der Statuszeile am unteren Window-Rand angezeigt. Die Kenntnis dieser Arbeitsweise von ED ist wichtig, da sie von der anderer Editoren, die zumindest TAB- und ESC-Zeichen »durchlassen«, abweicht. Man kann ED daher als »Filter« für Textdateien fremder Editoren verwenden, die ihre Zeilen z.B. mit Nullbytes oder einfachem Return abschließen, oder den Text mit nicht-normgemäßen ESC-Sequenzen verunzieren.

Die Bedienung

Nach dem Starten präsentiert sich ED mit einem Ihren Wünschen entsprechenden Console-Window auf dem Workbench-Screen (Bild). Falls das angegebene Textfile geladen werden konnte, wird der Anfang des Textes in dem Window angezeigt, andernfalls ist es leer.

Mit den Cursortasten kann wie gewohnt der Cursor im gesamten Window bewegt werden. Darüber hinaus positioniert ein Mausklick ins Window den Cursor an die Position des Mauszeigers. Neu eingegebene Zeichen werden immer an der Cursorposition in den Text eingefügt (Insert Mode), wobei evtl. rechts davon stehende Zeichen der Zeile weiter nach rechts verschoben werden. Stößt der Cursor rechts (oder links) an den Rand des Windows, so wird der Bildschirm in 10er Schritten horizontal gescrollt. Durch Drücken der Return-Taste wird die aktuelle Zeile an der Cursorposition aufgespalten (Split) und der vorher rechts vom Cursor befindliche Teil der Zeile in einer neu eingefügten Zeile dargestellt.

Zu Beginn einer neuen Zeile wird der Cursor an die Spalte bewegt, die durch den linken Rand (Left Margin) z.B durch den Befehl »SL« (s.u.) festgelegt wurde. Normalerweise ist das die erste Spalte.

Bewegt sich der Cursor bei Eingabe eines neuen Textes über den rechten Rand (Right Margin), der z.B durch den Befehl »SR« oder das Schlüsselwort »WIDTH« beim Start festgelegt wurde, hinaus, wird automatisch eine neue Zeile begonnen und ggf. das letzte Wort der alten Zeile vollständig in die neue Zeile übernommen (Wordwrap). Dieser Wordwrap kann durch den Befehl »EX« bei Bedarf für die aktuelle Zeile außer Kraft gesetzt werden.

Die unmittelbaren Befehle

Einige Tasten sind bei ED mit besonderen Funktionen ausgestattet. Weil sie sofort durch Drücken einer Taste (oder -sequenz) ausgeführt werden, spricht man auch von »unmittelbaren Befehlen« (Immediate Commands). Tabelle 1 enthält die Default-Belegungen aller Tasten,

Tabelle 1: Die Default-Belegung aller redefinierbaren Tasten.

Nr.	Token	Belegung	Taste(n)	Nr.	Token	Belegung	Taste(n)	Nr.	Token	Belegung	Taste(n)
1	1	"	F1	20	20	"	<Shift> F10	39	^M	S	<Ctrl> M , <Return> , <Enter>
2	2	"	F2	21	21	CS	<Shift> <Cursor_Left>	40	^N	"	<Ctrl> N
3	3	"	F3	22	22	CE	<Shift> <Cursor_Right>	41	^O	DW	<Ctrl> O
4	4	"	F4	23	23	T	<Shift> <Cursor_Up>	42	^P	"	<Ctrl> P
5	5	"	F5	24	24	B	<Shift> <Cursor_Down>	43	^Q	"	<Ctrl> Q
6	6	"	F6	25	25	DC	<Delete>	44	^R	WP	<Ctrl> R
7	7	"	F7	26	^@	BS	<Ctrl> <BackTick (`)>	45	^S	"	<Ctrl> S
8	8	"	F8	27	^A	A //	<Ctrl> A	46	^T	WN	<Ctrl> T
9	9	"	F9	28	^B	D	<Ctrl> B	47	^U	PU	<Ctrl> U
10	10	"	F10	29	^C	"	<Ctrl> C	48	^V	VW	<Ctrl> V
11	11	"	<Shift> F1	30	^D	PD	<Ctrl> D	49	^W	"	<Ctrl> W
12	12	"	<Shift> F2	31	^E	EP	<Ctrl> E	50	^X	"	<Ctrl> X
13	13	"	<Shift> F3	32	^F	FC	<Ctrl> F	51	^Y	EL	<Ctrl> Y
14	14	"	<Shift> F4	33	^G	RE	<Ctrl> G	52	^Z	"	<Ctrl> Z
15	15	"	<Shift> F5	34	^H	DL	<Ctrl> H , <Backstep>	53	^[	CM	<Ctrl> [, <ESC> , <Ctrl> Ü
16	16	"	<Shift> F6	35	^I	TB	<Ctrl> I , <Tab>	54	^\	"	<Ctrl> \
17	17	"	<Shift> F7	36	^J	"	<Ctrl> J	55	^]	CT	<Ctrl>] , <Ctrl> +
18	18	"	<Shift> F8	37	^K	"	<Ctrl> K	56	^^	"	<Ctrl> 6 , <Ctrl> #
19	19	"	<Shift> F9	38	^L	"	<Ctrl> L	57	^_	"	<Ctrl> - , <Ctrl> ß

wobei einige normalerweise noch unbelegt sind. Die Bedeutung der Buchstaben der Spalte »Belegung« ist identisch mit den in Tabelle 2 erklärten Extended Commands. Beachten Sie, daß Sie die Belegung der Tasten beliebig redefinieren können (s.u.).

Die erweiterten Befehle

Neben diesen unmittelbaren Befehlen beherrscht ED auch noch sogenannte »erweiterte Befehle« (Extended Commands): Nach Drücken der Escape-Taste springt der Cursor in die Statuszeile am unteren Rand des ED-Windows. Hier läßt sich nun hinter dem Asterisk (»*«) ein »erweiterter Befehl« eingeben, wodurch die Leistungsfähigkeit von ED ganz erheblich gesteigert werden kann. Die Statuszeile wird deshalb auch oft als Befehlszeile (Command Line) bezeichnet. Durch Drücken der Return-Taste wird der Befehl ausgeführt, und evtl. eine Fehlermeldung an der gleichen Stelle (also in der Statuszeile) ausgegeben. Anschließend springt der Cursor wieder in den Textbereich. Beachten Sie bitte, daß in der Statuszeile die Cursortasten und die »Del«-Taste nicht wie gewohnt funktionieren.

Ein erweiterter Befehl besteht aus einem ein oder zwei Buchstaben langen Befehlswort und ggf. mehreren Argumenten. z.B.

```
T
A/Hallo/
SF 1 'E//AMIGA-Magazin/'
```

Damit ED eindeutig zwischen Befehlsworten und Zeichenketten (Strings), in denen zufällig die gleichen Buchstaben vorkommen, unterscheiden kann, müssen Strings (also z.B. Texte, nach denen gesucht werden soll, oder Filenamen) an beiden Seiten mit demselben Begrenzungszeichen umrahmt werden. Als Begrenzungszeichen sind alle Zeichen zugelassen, die nicht im einzurahmenden String vorkommen, und keines von folgenden Zeichen sind: Ziffern, Buchstaben, Leerzeichen, Fragezeichen, Semikolon, runde Klammern. Begrenzungszeichen sind z.B. »/!\:'«.

Erlaubte Befehle wären z.B.:

```
OP /s:Startup-Sequence/
SA !df0:s/Startup-Sequence!
E  /Computer/AMIGA/
```

Im zweiten Beispiel durfte als Begrenzungszeichen kein Schrägstrich verwendet werden, da er bereits im Filenamen vorkommt. Tabelle 2 enthält eine Übersicht über alle erweiterten Befehle, die ED beherrscht. Die meisten Befehle sind anhand der Tabelle – notfalls mit etwas »Trial and Failure«, sprich Ausprobieren – selbsterklärend. Die Spalte »Mnemonic« enthält kurze englische Merkwörter, von deren Anfangsbuchstaben sich in der Regel der Name eines Befehls herleitet.

Die Befehlswörter können in Groß- oder Kleinschrift (auch gemischt) verwendet werden. Man kann auch mehrere Befehle gleichzeitig in der Befehlszeile eingeben. Sie werden dann durch Semikola »;« voneinander getrennt. Darüber hinaus kann vor jeden Befehl eine Zahl geschrieben werden, die angibt, wie oft der Befehl ausgeführt werden soll. Der spezielle Befehl »RP« (Repeat) wiederholt den

Tabelle 2: Alle Befehle, die ED beherrscht

Befehl	Mnemonic	Funktion
A/s/	Append	fügt »s« nach der aktuellen Zeile ein
B	Bottom	springt ans Textende
BE	Block End	setzt das Blockende
BF	Backward Find	sucht rückwärts
BS	Block Start	setzt den Blockanfang
CE	Cursor End	setzt Cursor an Zeilenende
CL	Cursor Left	bewegt Cursor ein Zeichen nach links
CM	Command	Extended Command (ESC)
CR	Cursor Right	bewegt Cursor ein Zeichen nach rechts
CS	Cursor Start	setzt Cursor an Zeilenanfang
CT	Cursor Toggle	wechselt zwischen Zeilenanfang/-Ende
D	Delete	löscht Cursorzeile
DB	Delete Block	löscht Block
DC	Delete Char	löscht Zeichen unter Cursor (Del-Taste)
DF	Describe FN Key	zeigt Belegung einer Funktionstaste an
DL	Delete Left	löscht Zeichen links vom Cursor (Backstep-Taste)
DW	Delete Word	löscht Wort unter Cursor
E/s/t/	Exchange	sucht nächstes »s« und ersetzt durch »t«
E	Exchange	sucht und ersetzt weiter, wie beim obigen Befehl
EL	Erase Line	löscht Zeile ab Cursor
EM	Evaluate Menus	berechnet neue Darstellung der Menüs
EP	Exchange Page	wechselt Cursor zwischen Seitenanfang/-Ende
EQ	Exchange Query	Suchen und Ersetzen nach Abfrage
EX	Extend Margins	Randauflösung: setzt Randeinstellung außer Kraft
F/s/	Find	sucht vorwärts nach »s«
F	Find	sucht vorwärts weiter, wie beim obigen Befehl
FC	Flip Case	wechselt ein Zeichen zwischen Groß-/Kleinschrift
FR	File Requester	schaltet File-Requester aus (an = FR1)
I/s/	Insert	fügt »s« vor der aktuellen Zeile ein
IB	Insert Block	kopiert Block an Cursorposition
IF	Insert File	File nach aktueller Zeile einfügen
J	Join Lines	hängt die nachfolgende Zeile ans Ende der aktuellen
LC	Lower Case	Case Sensitive: Groß-/Kleinschreibung beachten
Mn	Move	springt an Anfang von Zeile »n«
N	Next	bewegt Cursor an Anfang der nächsten Zeile
NW	New	löscht Puffer nach Sicherheitsabfrage
OP	Open File	lädt neues Textfile
P	Previous	bewegt Cursor an Anfang der vorherigen Zeile
PD	Page Down	blättert eine (halbe) Seite vorwärts
PU	Page Up	blättert eine (halbe) Seite rückwärts
Q	Quit	Beenden ohne Speichern, aber mit Sicherheitsabfrage
RE	Repeat Extended	wiederholt letzten erweiterten Befehl (ESC)
RF	Run File	führt Script-File mit Macro-Befehlen aus
RK	Reset Keys	Sondertasten-Belegung normalisieren
RP	Repeat	solange wiederholen, bis Fehler oder Textende
RV	Request Values	sendet aktuelle Werte von ED an ARexx
RX	Rexx	führt ARexx-Befehl oder ARexx-Script-File aus
S	Split	spaltet aktuelle Zeile an Cursorposition auf
SA	Save As	speichert Text unter wählbarem Namen ab
SB	Show Block	zeigt markierten Block an
SF	Set FN Key	Funktionstasten belegen
SH	Show	zeigt Informationen an
SI	Set Item	Menü-Item definieren
SLn	Set L Margin	setzt linken Rand auf Spalte »n«
SM	Send Message	Message ausgeben
SRn	Set R Margin	setzt rechten Rand auf Spalte »n«
STn	Set Tab	setzt Tabulator-Abstände auf »n« Zeichen
T	Top	springt an Textanfang
TB	Tab	springt nächste Tabulator-Position an
U	Undo	Änderungen in aktueller Zeile rückgängig machen
UC	Upper Case	Ignore Case: Groß-/Kleinschreibung ignorieren
VW	Validate Window	ED-Fenster neuschreiben
WB	Write Block	speichert Block in ein File
WN	Word Next	bewegt Cursor ein Wort vorwärts
WP	Word Previous	bewegt Cursor ein Wort rückwärts
X	Exit	Beenden mit Speichern und Sicherheitskopie
XQ	Exit Query	Speichern und Beenden nach Abfrage

nachfolgenden Befehl, bis ein Fehler auftritt (z.B. weil das Textende erreicht wurde). Durch Drücken einer beliebigen Taste kann notfalls die Ausführung wiederholter Befehle jederzeit unterbrochen werden. Schließlich kann man noch durch runde Klammern mehrere Befehle in der Befehlszeile zu Gruppen zusammenfassen, und mit eigenen Wiederholungsfaktoren versehen. Bei Befehlen, die mindestens ein Argument erfordern, können Sie dieses Argument auch vom Anwender durch eine Frage mit frei wählbarem Text erfragen. Dazu dient ein Fragezeichen vor dem mit Begrenzungszeichen umrahmten Fragetext (maximal 29 Zeichen). Hierzu zeigt die Tabelle ein paar Beispiele:

Befehlsbeispiele

Befehl	Funktion
M ?/Zeile:/ 5 CR M 8;CS;9 CR LC;M 12;5 E/Amiga/AMIGA/ T;RP(E// /;N) T;RP(E// /;CS;DW;N);VW T;RP(CE;E// /;CL;DW;N);VW	Fragt nach »Zeile:« und bewegt Cursor in die angegebene Zeile. bewegt den Cursor 5 Spa-lten nach rechts. bewegt Cursor in Spalte zehn von Zeile acht. ersetzt ab Zeile 12 fünf mal das Wort »Amiga«durch »AMIGA«. rückt den Text zwei Spalten weiter nach rechts. löscht alle Leerzeichen am Zeilenanfang (im gesamten Text). löscht überflüssige Leerzeichen am Zeilenende
10(CS;3 WN;30 E// /; CS;29 CR;DW;N);VW	rückt (z.B. bei einer Tabelle) in den folgenden zehn Zeilen das jeweils dritte Wort an Spalte 30.
SA/NIL:/;OP/lvo.i/;LC;RP (CS;E/ equ/ =/;CL;30 E// /;CS;29 CR;DW;VW;N)	lädt das File »lvo.i« und ersetzt alle »equ«-Befehle durch Gleichheitszeichen. Außerdem sollen alle Gleichheitszeichen untereinander in Spalte 30 stehen.

Beachten Sie, daß ED bei Wiederholungen von Befehlen nicht immer von selbst nach jedem Befehl den kompletten Bildschirm aktualisiert, um Zeit zu sparen. Daher sieht es während der Bearbeitung mancher Befehle so aus, als würden sie nicht richtig funktionieren. Durch Verwendung zusätzlicher »VW«-Befehle können Sie natürlich (auf Kosten von Bearbeitungszeit) ED jederzeit anweisen, den Bildschirm komplett zu aktualisieren.

Noch ein nützlicher Tip: Die Eingabe der erweiterten Befehle ist zuweilen recht mühselig. Da es sich beim ED-Window um ein Console-Window handelt, können Sie sich durch geschickte Benutzung des Console-Clippings (Markieren und Einfügen mit der Maus) sehr viel Arbeit ersparen. Schreiben Sie einfach den betreffenden erweiterten Befehl in ein Shell-Window und kopieren Sie ihn von dort bei Bedarf in die Befehlszeile von ED. Wenn Sie als erstes Zeichen im Shell-Window ein Semikolon eingeben (aber dieses bitte nicht mit kopieren!), können Sie im Shell-Window auch »Return« drücken, ohne eine Fehlermeldung zu erhalten, und so bequem die Command-Line-History (Durchblättern alter Befehle mit Cursor_Up-Taste) benutzen. <Ctrl G> führt den letzten erweiterten Befehl nochmal aus.

Die Funktionstasten

ED erlaubt es, die Funktionstasten zu belegen. Ferner können fast alle Sondertasten (ESC, Return, Tab, usw.) und Control-Tastensequenzen beliebig redefiniert werden. Das Drücken einer solchen Taste (oder Tastensequenz) wirkt so, als hätten Sie die ESC-Taste gedrückt und würden dann die, mit der gedrückten Funktionstaste verbundenen, Befehle in der Statuszeile eingeben und ausführen. Die Definition einer Funktionstaste (oder einer Sondertaste oder Ctrl-Sequenz) erfolgt durch den Befehl »SF« (Set FN Key), der folgende Syntax hat:

```
SF [Nummer|Token] <Belegung>
```

Dem Befehlswort »SF« folgt wahlweise die Nummer der zu belegenden Taste gemäß Tabelle 1, oder ihr sogenanntes »Token«, das sich in der Regel aus der Aufschrift der Taste herleitet, und ebenfalls der Tabelle entnommen werden kann. Danach folgt die in Begrenzungszeichen eingerahmte neue Belegung der Taste. Zum Beispiel:

```
SF ^C 'F/Amiga/'
```

bewirkt, daß nach jedem Drücken von <Ctrl C> der Cursor auf die nächste Textstelle, die das Wort »Amiga« enthält, positioniert wird. Man hätte genausogut »SF 29 'F/Amiga/'« verwenden können, denn die Sequenz »<Ctrl C> hat die Positionsnummer 29 in der Tabelle.

```
SF 01 'E//AMIGA/'
```

bewirkt z.B. daß bei jedem Drücken von F1 der Text »AMIGA« an der aktuellen Cursorposition eingefügt wird. Natürlich können Sie auch komplette Befehlszeilen mit erweiterten Befehlen auf die Funktionstasten, Sondertasten oder Ctrl-Tastensequenzen legen, die dann durch einen einzigen Tastendruck ausgeführt werden, und z.B. auch Macro-Scriptfiles oder ARexx-Programme starten könnten (s.u.).

Die Menüs

ED bietet freidefinierbare Roll-Down-Menüs. Normalerweise werden die Menüs durch das Macro-Scriptfile beim Programmstart definiert, können aber auch jederzeit geändert werden. Um zu verstehen, wie diese Menüs definiert werden, tippen Sie bitte folgende 5 Zeilen nacheinander wie erweiterte Befehle ein: Drücken Sie für jede Zeile die ESC-Taste und geben dann in der Statuszeile hinter dem »*« eine Zeile ein. Beenden Sie jede Zeile mit der Return- Taste.

```
SI 0 1 'Mein Menü'
SI 1 2 'Anfang' 'T'
SI 2 2 'Ende' 'B'
SI 3 0
EM
```

Wenn Sie nun die rechte Maustaste drücken, werden Sie feststellen, daß Ihr ED-Window jetzt ein Menü mit dem Titel »Mein Menü« und den beiden Menüpunkten »Anfang« und »Ende« besitzt. Doch damit nicht genug: Bei Auswahl von »Anfang« bewegt sich der Cursor - natürlich an den Textanfang. Bei »Ende« ans Textende.

Jede Menüdefinition besteht aus einem oder mehreren »SI«-Befehlen (Set Item) mit ihren jeweiligen Parametern, und dem abschließenden »EM«-Befehl.

»EM« (Evaluate Menus) berechnet das komplette Menü (also nicht nur die letzten Änderungen) neu und verbindet es dann mit dem ED-Window. Bitte führen Sie den »EM«-Befehl niemals aus, wenn Sie nicht wirklich sicher sind, daß alle vorhergehenden »SI«-Befehle konsistent (untereinander sinnvoll) sind.

Formal sieht ein »SI«-Befehl so aus:

```
SI <Nummer> <Kennung> <Argumente>
```

Dabei ist <Nummer> die laufende Nummer des Menü-Eintrags (Item), der definiert werden soll (maximal 99). Es müssen alle Menü-Items von 0 bis <Nummer>-1 bereits existieren. Sinnvollerweise sollte man Items in aufsteigender Reihenfolge definieren. Die Menü-Items werden in der Reihenfolge ihrer Nummer im späteren Roll-Down-Menü dargestellt. Also Menüpunkt Nummer 2 unter Menüpunkt Nummer 1, und ein Menü-Titel mit z.B. Nummer 3 in der obersten Zeile, aber rechts von Menüpunkt Nummer 2.

Leerzeichen zwischen »SI« und <Nummer> sind optional (können also auch entfallen), zwischen <Nummer> und <Kennung> muß aber mindestens ein Leerzeichen stehen. <Argumente> werden, sofern sie nötig sind, jeweils durch Begrenzungszeichen umrahmt und ggf. durch mindestens ein Leerzeichen voneinander getrennt.

<Kennung> bezeichnet die Art des Menü-Eintrags und bestimmt dadurch auch Art und Anzahl eventueller Argumente.

Es gibt folgende fünf Menü-Kennungen:

❒ Kennung 1 definiert einen Menütitel.

Das einzige erlaubte Argument ist der Text des Titels, umrahmt von Begrenzungszeichen. Das Menü-Item mit der Nummer 0 muß immer ein Menütitel sein. Z.B.: SI 0 1 "Project"

❒ Kennung 2 definiert einen Menüpunkt.

Wird dieser Menüpunkt ausgewählt, indem die rechte Maustaste losgelassen wird, während sich der Mauszeiger über diesem Menüpunkt befindet, wird der durch das zweite Argument bezeichnete erweiterte Befehl ausgeführt. Das erste Argument ist der Name des Menüpunktes, umrahmt von Begrenzungszeichen. Das zweite Argument wird so ausgeführt, als hätten Sie die ESC-Taste gedrückt und dann in der Statuszeile das zweite Argument als Befehl eingegeben. Selbstverständlich, kann dieser Befehl auch aus mehreren erweiterten Befehlen, durch Semikola (»;«) getrennt, bestehen und Wiederholungsfaktoren enthalten. Z.B. läßt sich der in vielen anderen Editoren vorhandene Befehl »Clone Line«, der die aktuelle Zeile vervielfältigt, so realisieren:

```
SI 1 2 'Clone Line' 'BS;BE;IB;N'
```

❒ Kennung 3 definiert ein Untermenü.

Dieser Item-Typ wird in der aktuellen Version von ED bislang nur optisch unterstützt und erlaubt als einziges Argument den mit Begrenzungszeichen umrahmten Namen.

❒ Kennung 4 definiert eine Trennlinie.

Anstelle des Menü-Items erscheint eine breite gerasterte Linie, die nicht ausgewählt werden kann. Dieses Item erfordert kein Argument.

❒ Kennung 0 definiert das logische Ende des gesamten Menüs.

Bei der Berechnung des Menüs durch »EM« ordnet der Amiga alle Menü-Items nach Nummern und versucht, das Layout festzulegen, bis ein Item mit der Kennung 0 gefunden wird. Jede Menüdefinition muß ein solches Item enthalten! Dieses Item erlaubt kein Argument.

Die Diskette zum Heft enthält mehrere Beispielmenüs und Funktionstastenbelegungen.

Die Macro-Files

Macro-Files sind Text-Dateien, die Befehle für ED enthalten. Alle Zeilen des Files werden dabei nacheinander so ausgeführt, als hätten Sie diese nach Drücken der ESC-Taste in der Statuszeile von ED als Befehle eingegeben.

Ein Macro-Scriptfile wird durch den »RF«-Befehl (Run File) gestartet, der als einziges Argument einen, wie immer mit Begrenzungszeichen umrahmten, Filenamen verlangt. Häufig enthalten Macro-Scriptfiles Befehlszeilen, die zu lang, zu zahlreich, oder zu unübersichtlich wären, um sie direkt auf Funktionstasten oder Menüs zu legen, oder direkt einzugeben. Ein konkretes Beispiel wären die oben besprochenen Menüdefinitionen, oder auch Funktionstastenbelegungen. Z.B. lädt »RF/S:Ed-Startup/« das normale Konfigurationsfile, welches die bekannten spärlichen 3 Menüs »Project«, »Movement« und »Edit« definiert.

Der ARexx-Port

Nach dem Start richtet ED einen öffentlichen Message-Port ein, über den das Programm mit ARexx kommunizieren kann. Dazu prüft das Programm, ob es bereits einen Message-Port namens »Ed« gibt. Falls nicht, erhält der Message-Port diesen Namen (»Ed«). Sollte der Message-Port schon existieren, wird nach »Ed_1« gesucht, danach nach »Ed_2« usw.

Im folgenden nehmen wir an, daß der RexxMaster-Prozess bereits gestartet wurde (z.B. durch Doppelklick auf das »RexxMast«-Icon im Verzeichnis »System« Ihrer Workbench-Diskette) und der Message-Port »Ed« heißt.

Bitte beachten Sie: Es gibt kein Protokoll, um exklusiven Zugriff auf ED zu erlangen. D.h., theoretisch können Sie und ARexx gleichzeitig an ein und demselben ED arbeiten. Das kann zu unerwünschten Veränderungen des Textes führen. Führen Sie deshalb bitte keine Operationen an ED aus (Text eingeben, Cursor bewegen, Mausklicks ins ED-Window, etc.), während ein ARexx-Script abgearbeitet wird (das denselben ED bedienen soll), und versuchen Sie nicht, von mehreren unterschiedlichen ARexx-Scripts gleichzeitig auf ED zuzugreifen.

```
/* DA.ed (Date) */
Address "Ed"
Datum=Translate(Date("E"),".","/")
'E!!'||Datum||'!'
```

Listing 1: »DA.ed« – Datum im Text einfügen auf Tastendruck

```
/* HDB.rexx */
Address "HDBACKUP_CBM"
options RESULTS
'STATUS f'
Merker = RESULT
parse var Merker fa fb fc fd ff
say "Files:" fc"/"fa
say " Size:" fd"/"fb
```

Listing 2: Auch »HDBackup« hat einen ARexx-Port

```
/* UL.ed (Upper Line) */
Address "Ed"
'RV/EdValues/'
Zeile=EdValues.CURRENT
Zeile=Delimit(Upper(Zeile))
'CS;EL'
'E'||Left(Zeile,1)||Zeile
'N'
EXIT
/* String begrenzen */
Delimit: Procedure
 Parse arg String
 dc="/\!:|§$%&=+*^-<>`'~"
 x0=Verify(dc,String)
 c1=Substr(dc,x0,1)
 String=c1||String||c1
return String
```

Listing 3: »UL« macht alle Buchstaben einer Zeile zu Großbuchstaben.

```
/* FL.ed (Fill Line) */
Address "Ed"
'RV/EdValues/'
Zeile=Trim(EdValues.CURRENT)
ri=EdValues.RIGHT
w=Words(Zeile)-1
sp=ri-Length(Zeile)-1
do
 if sp<1 | w<1 then Break
 ta=trunc(sp/w)
 Rest=sp-ta*w
 do n=1 for w
   a=wordindex(Zeile,n+1)
   if a=0 then Break
   nn=ta+(Rest>0)
   Zeile=insert(" ",Zeile,a-1,nn)
   Rest=Rest-1
 end
 Zeile=Delimit(Zeile)
 'CS;EL'
 'E'||Left(Zeile,1)||Zeile
end
'N'
EXIT
/* String begrenzen */
Delimit: Procedure
 Parse arg String
 dc="/\!:|§$%&=+*^-<>`'~"
 x0=Verify(dc,String)
 c1=Substr(dc,x0,1)
 String=c1||String||c1
return String
```

Listing 4: »FL« zentriert eine Zeile rechtsbündig

Grundsätzlich läßt sich jeder erweiterte Befehl, der über die Tastatur eingegeben werden kann, auch durch ARexx ausführen. Zusammen mit den leistungsfähigen Stringfunktionen von ARexx und dessen strukturierter Programmierbarkeit (Schleifen, Funktionen, etc.) eröffnen sich wirklich ungeahnte Möglichkeiten der Textbearbeitung. Das folgende ARexx-Script würde z.B. den Namen einer sehr bekannten Computerzeitschrift in das ED-Window schreiben.

```
/* Hallo ED! */
Address "Ed"
"A/AMIGA-Magazin/"
```

Die erste Zeile des Programms ist die obligatorische Kommentarzeile, mit der alle ARexx-Programme beginnen müssen. Mit »Address "Ed"« wird der Messageport festgelegt, an den ARexx alle Befehle weitersendet, die keine ARexx-Befehle sind (sog. externe Kommandos). Die Zeile »"A/AMIGA-Magazin/"« schließlich ist der Befehl, den ARexx an ED sendet. Die Zeile ist (sicherheitshalber) in Anführungszeichen geschrieben, weil ARexx sonst versuchen könnte, den Befehl selbst auszuführen, statt ihn unverändert an ED weiterzuleiten. Wie Sie sehen, dienen zum Fernbedienen von ED durch ARexx in einem ARexx-Script genau die Befehle, die Sie als erweiterte Befehle in der Statuszeile von ED eingeben würden, um dieselbe Funktion zu erreichen.

Nach Ausführung jedes Befehls, den ARexx an ED sendet, erhält ARexx einen Return-Code zurückgemeldet, der im allgemeinen Aufschluß über die ordnungsgemäße Bearbeitung des letzten Befehls gibt. Dieser Return-Code wird in der ARexx-Standardvariablen »RC« übergeben, die genaue Bedeutung kann Tabelle 3 entnommen werden. Bitte beachten Sie, daß Return-Codes von 10 und größer normalerweise das ARexx-Script abbrechen, wenn nicht ein anderer FAILAT-Level gewählt wurde, oder die Fehler intern (z.B. durch »SIGNAL ON FAILURE«) abgefangen werden.

ARexx-Scripts für ED können auf drei verschiedene Weisen gestartet werden:

Extern, durch Eingabe des Befehls »RX« gefolgt vom Namen des ARexx-Scriptfiles in einem CLI/Shell-Window. Explizit, durch Eingabe des erweiterten ED-Befehls »RX« gefolgt von dem mit Begrenzungszeichen umrahmten Filenamen, bzw. Auswahl des entsprechenden Menüpunktes. Am interessantesten ist jedoch die implizite Methode:

Wenn ED einen unbekannten, erweiterten Befehl ausführen soll, wird der Befehl (sofern ED ihn nicht wegen grober Syntax-Fehler reklamiert) an ARexx gesendet. ARexx versucht dann, ein ARexx-Scriptfile zu laden und auszuführen, das unter folgenden Namen in dieser Reihenfolge gesucht wird (angenommen, der eingegebene ED-Befehl sei
`»DA«): »REXX:DA.ed«,»REXX:DA«,»DA.ed«,»DA«`

Haben Sie z.B. Listing 1 abgetippt, und unter dem Namen »DA.ed« im Verzeichnis »REXX:« (normalerweise ist das »SYS:s«) gespeichert, so würde jetzt das aktuelle Datum (in deutscher Schreibweise!) an der Cursorposition in Ihren Text eingefügt.

```
/* FT.ed (Format Text) */
Text="Formatting Text...",
||"0a"x||"Ctrl-C to Quit!"
NI="1b"x||"[8m"
Window="CON:416/0/178/30/",
||"Ed Message/nosize/smart"
call Open("Wd",Window)
call Writech("Wd",Text||NI)
Address "Ed"
do forever
 '12SL1'
 'RV/EdValues/'
 Zeile=EdValues.CURRENT
 ri=EdValues.RIGHT-1
 sp=ri-Length(Zeile)
 Select
  when sp<0 then
   do
    'CS'
    a=Lastpos(" ",Zeile,ri)
    if a<1 then a=ri-1
    a||'CR;10SL1'
    'S'
   end
  when sp>1 then
   do
    'CS;CT;5SL1'
    if Right(Zeile,1)~=" "
      then 'E// /'
    'J'
   end
  Otherwise
   'N;10SL1'
 end
end
```

Listing 5: Der »FT«-Befehl formatiert den Text

Sie können sich also auf diese Weise beliebig viele zusätzliche Befehle (»Genies«) erzeugen. Bei diesen selbsterzeugten Befehlen wird der Rest der Befehlszeile immer ignoriert, und kann daher zur Übergabe von Parametern an den Befehl dienen (»LASTCMD«, s.u.). Bitte beachten Sie: ARexx-Scripts werden immer asynchron gestartet (beliebig viele könnten gleichzeitig laufen). Versehen Sie einen selbsterzeugten ED-Befehl daher niemals mit einem Wiederholungsfaktor, und warten Sie, bis das letzte ARexx-Script vollständig abgearbeitet wurde, bevor Sie ein neues starten; Sie würden ED sonst hoffnungslos verwirren!

Um wirklich interaktiv mit ED zu kommunizieren, genügt es nicht, ED lediglich Befehle zu erteilen, und deren korrekte Bearbeitung anhand des Return-Codes (RC) zu prüfen. Deshalb können Sie von ED auch weitere Informationen über Cursorposition, Zeilennummer, Randeinstellungen, ... und natürlich den vollständigen Inhalt (Textstring) der aktuellen Zeile, anfordern. Die Art der Datenübertragung weicht bei ED von der sonst üblichen Methode ab: Normalerweise fordert man mit einem speziellen Befehl einen erwünschten Wert an, und erhält diesen dann in der ARexx- Standard-Variablen »RESULT«.

Übrigens: Wußten Sie, daß auch das Festplatten-Backup-Programm »HDBackup« auf Ihrer Extras-Diskette einen ARexx-Port hat? Listing 2 zeigt z.B., wie man »HDBackup« Informationen über Anzahl und Länge der zum Backup selektierten Files entlockt.

Bei ED ist das (für den Anwender) etwas bequemer gelöst: Es gibt nur einen einzigen Befehl (»RV«), der gleich alle Daten auf einmal anfordert. Als einziges Argument verlangt der »RV«-Befehl (Request Values) den, wie gewohnt mit Begrenzungszeichen umrahmten, Namen eines Variablen- Feldes, dem die Daten zugewiesen werden sollen. Z.B.: »"RV/EdValues/"« Anschließend enthält dann »EdValues.X« die X-Position des Cursors, »EdValues.LINE« die aktuelle Zeilennummer, usw. Insgesamt werden durch den einen Befehl 15 Elemente des Feldes »EdValues« definiert. Wie Sie sehen, verwendet ED als Indizes nicht irgendwelche nichtssagenden Zahlen (z.B. 1-15), sondern Mnemonics, aus denen man leicht die Bedeutung erkennen kann. Die Indizes haben folgende Namen und Bedeutungen:

.X: X-Position des Cursors im ED-Window. Nicht unbedingt identisch mit der Spaltenposition in der aktuellen Textzeile (dazu müßte man .BASE addieren)!

.Y: Y-Position des Cursors im ED-Window. Diese ist nicht unbedingt identisch mit der Zeilennummer der aktuellen Zeile (die steht »fest« in .LINE)!

```
/* CU.ed (Cursor) */
Address "Ed"
NI="1b"x||"[8m"
Wind="CON:444/0/150/30/Cursor",
||"/inactive/close/nosize/wait"
call Open("Wd",Wind)
'RV/EdValues/'
YP=EdValues.Y
LL=EdValues.LINE
LM=EdValues.LMAX
LI=LL
if LL>LM+1 then LI=LI+1
IX=EdValues.X
CO=EdValues.BASE+IX
TE="";if LL<LM+2 then TE="~"
TE="Line:"||TE||LI" Col:"CO,
 ||"0a"x"Y:"YP "  X:"IX
call Writech("Wd",TE||NI)
```

Listing 6: »CU«: Cursorposition im Window

.BASE: Bekanntlich scrollt das ED-Window horizontal, falls der Cursor über die Spaltenposition, die maximal im Window dargestellt werden kann, hinausbewegt wird. .BASE gibt dann den Offset an, also die Anzahl der Zeichen, die am linken Windowrand nicht mehr dargestellt werden. (Aufgrund der verwendeten Scrollmethode immer 0 oder ein ganzzahliges Vielfaches von 10.)

.LINE: Zeilennummer der aktuellen Zeile innerhalb des gesamten Textes. Achtung, Bug: Dieser Wert stimmt nur für die erste Seite, wenn Textzeile 1 in der ersten Windowzeile steht, sonst ist er immer um 1 zu klein.

.LEFT: Left Margin. Linker Rand, eingestellt mit »SL«. Bestimmt die Position, an die der Cursor beim Beginnen einer neuen Zeile (z.B. nach Return) positioniert wird. Normalerweise 1.

.RIGHT: Right Margin. Rechter Rand, eingestellt mit »SR« oder »WIDTH« beim Aufruf. Bestimmt die Position, an der bei neueingegebenen Zeilen ein automatischer Wordwrap erfolgt. Normalerweise .WIDTH+1.

.TABSTOP: Tabulator-Abstände, eingestellt mit »ST« oder »TABS« beim Aufruf. Defaultmäßig 3.

.LMAX: Anzahl der Textzeilen, die gleichzeitig ins ED-Window passen, minus 1.

.WIDTH: Anzahl der Zeichen, die in einer Zeile des ED-Windows dargestellt werden können, minus 1.

.EXTEND: Boolean-Wert der Randauflösung (»EX«) für die aktuelle Zeile. Normalerweise 0 für keine Randauflösung, d.h. beim Überschreiten von .RIGHT tritt ein Wordwrap ein.

.FORCECASE: Boolean-Wert der Case-Sensitivity bei allen Such-Befehlen. Normalerweise 1 für »Groß-/Kleinschreibung beachten« (sensitive, »LC«).

.FILENAME: Name der Datei im Textpuffer.

.CURRENT: Inhalt der aktuellen Zeile.

.LASTCMD: Letzter erweiterter Befehl, der über die Tastatur eingegeben wurde (komplette Befehlszeile).

.SEARCH: letzter Suchstring.:

Der in Listing 3 abgedruckte Befehl »UL« (Upper Line) wandelt alle Buchstaben der aktuellen Zeile in Großbuchstaben um. »FL« (Fill Line) aus Listing 4 fügt in der aktuellen Zeile zusätzliche Leerzeichen ein, so daß der Text rechtsbündig ist. Listing 5 zeigt den Befehl »FT« (Format Text), der den Text ab der Cursorzeile so umformatiert, daß alle Zeilen optimal ausgefüllt werden (Zeilenumbruch). »CU« (Cursor) aus Listing 6 zeigt die Cursorposition in einem eigenen Window an.

Die Diskette zum Heft enthält noch einige weitere selbsterzeugte Befehle und Arexx-Listings zum Steuern von ED. ■

Literaturhinweise:

[1] Amiga Benutzerhandbuch (Lieferumf. jedes Amiga vor OS 2.0)

[2] Commodore-Amiga Inc.: Amiga-DOS-Handbuch, Markt & Technik Verlag, München, ISBN 3-89090-465-3

[3] Alexander Kochann und Oliver Reiff: Königliches Paar, AMIGA-Magazin 3/93, Seite 52 ff.

[4] Christian Kuhnert: Amiga Profi-Know-How, DATA BECKER, Düsseldorf, ISBN 3-89011-301-X

Tabelle 3: Fehlermeldungen

RC	Bedeutung
0	OK
1	Last line deleted
2	No room in buffer
3	Creating new file
4	Input lines truncated
5	Top of file
6	End of file
7	Line too long
8	Unknown command
9	Unmatched ()
10	Commands abandoned
11	Syntax error
12	Unable to open file
13	Number expected
14	No block marked
15	Cursor inside block
16	Block incorrectly specified
17	Search failed
18	Tabs in input file expanded
19	Rexx not available
20	Out of memory for operation
21	Unknown internal error

ARexx-Tips

Befehls-Baukasten

Eine der herausragenden Eigenschaften von ARexx ist es, daß man Anwenderprogramme quasi fernsteuern kann. Das läßt sich z.B. nutzen, um Programme wie CygnusED oder DirOpus um eigene Befehle zu erweitern. Der folgende Artikel zeigt, wie es gemacht wird.

von Kurt Schuster

Die zwei ARexx-Scripts »StorePath.dopus« und »RestorePath.dopus« erleichtern die Arbeit mit »Directory Opus« (kurz »DOpus«). Nach der Arbeit mit »DOpus« werden die Pfade beider Fenster gespeichert und bei einem Neustart wieder geladen. Man betritt seine Arbeitsfläche, so wie man sie verließ. Wenn man das nicht will, besteht die Möglichkeit, »DOpus« zu verlassen, auch ohne einen Pfad zu speichern.

```
/* StorePath.dopus - Speichert Pfad mit dem
   DOpus verlassen wird */
address 'dopus_rexx'
options results
filename = "s:DOpusDirStore"
status 13 0
path_0 = result
status 13 1
path_1 = result
sc=open(wdat,filename,'W')
if sc=0 then exit 10
writeln(wdat,path_0)
writeln(wdat,path_1)
sc=close(wdat)
quit
exit                © 1993 M&T
```

»StorePathdopus«: Speichert beim Verlassen von DOpus den Pfad

Gehen Sie wie folgt vor:

1. Kopieren Sie die beiden Dateien »StorePath.dopus« und »RestorePath.dopus« in den »REXX:«-Ordner
2. Konfigurieren Sie in »DOpus« ein Gadget, am besten über dem kleinen »Q«uit-Gadget in der rechten unteren Ecke des Bildschirms mit:

```
- Gadget name    QUIT+SAVE
- Funktion       StorePath
- (Funkt.art)    ARexx (rechts v. Qualifier)
- Stack size      4000
- (alle anderen Einstellungen sind nicht
  selektiert)
```

3. Konfigurieren Sie die Datei »System/ARexx/Startup script«: Tragen Sie dort »RestorePath« ein.
4. Löschen Sie »System/Directories/Auto left Directory« und »Auto right Directory«.

Und dann bleibt nur noch Ihnen viel Spaß mit dem neuen DirOpus zu wünschen – auf ähnliche Art können Sie natürlich auch andere eigene Verbesserungen einführen.

```
/* RestorePath.dopus
   Lädt Pfad mit dem DOpus verlassen wurde */
address 'dopus_rexx'
options results
filename = "s:DOpusDirStore"
sc=open(rdat,filename,'R')
if sc=0 then do
        path_0="Work:"
        path_1="RAM:"
end
else do
        path_0=readln(rdat)
        path_1=readln(rdat)
end
sc=close(rdat)
status 13 0 set path_0
status 13 1 set path_1
exit              © 1993 M&T
```

»ReStorePath.dopus«: Zum automatischen Setzen eines Pfades

Mit den ARexx-Scripts »StorePath.ced« und »RestorePath.ced« erhält auch der beliebte Editor »CygnusEd« die Fähigkeit, sich Dateinamen mit deren Pfaden zu merken. Der Anwender kann auf einfache Weise mit Hilfe des CED-File-Requesters aus den zuletzt bearbeiteten Dateien wählen, ohne sich um deren Pfade kümmern zu müssen. Öffnet man eine Datei, wird ihr Name automatisch der »Schnell-Auswahlliste« hinzugefügt; die Position des Cursors wird, falls die Datei schon einmal gesichert wurde, restauriert.

```
/* StorePath.ced QRT 100992 V0.8
   speichert Pfad und aktuelle Cursorposition
   einer Datei durch Amiga-w(rite), keine
    Werte speichern durch die analogen
    Menüaufruf (mit rechter Maustaste ...)
    oder Alt-w(rite)
*/
address 'rexx_ced'
options results
store_path = "s:cedstore/"
status 19; pun = result
status 21; name = result
status 56; nby = result;
status 46; nby = nby + result
sc=open(wdat,store_path||name,'W')
if sc=0 then exit
writeln(wdat,pun)
writeln(wdat,nby)
sc=close(wdat)
save
exit              © 1993 M&T
```

»StorePath.ced«: Speicherhilfe für den Editor CygnusEd-Professional

Wenn Sie ARexx ordnungsgemäß installiert haben, CED Version 2.12 besitzen und sich die Dateien »StorePath.ced«, »RestorePath. ced« »QuickOpenInst.rexx« und »QuickOpenInst. txt« (letztere aus Platzgründen nur auf PD-Diskette zum Heft) in einem beliebigen Verzeichnis oder auf Diskette befinden, tippen Sie vom CLI aus (nehmen wir an die vier Dateien befinden sich auf »Df0:QuiOp«):

```
rx DF0:QuiOp/QuickOpenInst
```

Unter WB 2.0 kann »rx« entfallen. Die Grundinstallation erfolgt nun per ARexx-Script, dann werden Sie noch aufgefordert die Makro-Installation »per Hand« (nach Anleitung auf PD-Diskette) durchzuführen und schon kann es losgehen. Das System läßt sich leicht erweitern, z.B. die CED-Funktionen »Quit« oder »Save as« könnte auch unterstützen werden. ■

```
/* RestorePath.ced QRT 100992 V0.8
   lädt ausgewählte Datei aus der Liste
   der zuletzt bearbeiteten Dateien, wenn
   Amiga-o(pen) benutzt wird, Auswahl aus
   beliebigem Verzeichnis durch die analogen
   Menüaufrufe (mit rechter Maustaste ...)
   oder Alt-o(pen)
*/
address 'rexx_ced'
options results
store_path = "s:cedstore/"
status 18; changes = result
if changes~=0 then do
        clear
        status 18; changes = result
        if changes~=0 then exit
end
q = '22'x
nby = 0
getfilename store_path
pun = result
if pun="RESULT" then exit
if index(pun,store_path)~=0 then do
        sc = open(rdat,pun,'R')
        if sc=0 then exit
        pun = readln(rdat)
        nby = readln(rdat)
        sc = close(rdat)
end
open q||pun||q
jump to byte nby
status 19; pun = result
status 21; name = result
sc=open(wdat,store_path||name,'W')
if sc=0 then exit
writeln(wdat,pun)
writeln(wdat,nby)
sc=close(wdat)
exit              © 1993 M&T
```

»RestorePath.ced«: Lädt Dateien immer aus richtigen Verzeichnis

3-D-Funktionsplotter

CIRCLE & Co für ARexx

ARexx wird erwachsen. Wir zeigen Ihnen, wie Sie mit ein paar Tricks dieser Sprache auch Grafikfunktionen beibringen; Funktionen, die alles in den Schatten stellen, was Amiga-Programmierer in Sachen 3-D-Grafik bisher mit den normalen AmigaBASIC-Befehlen versucht haben.

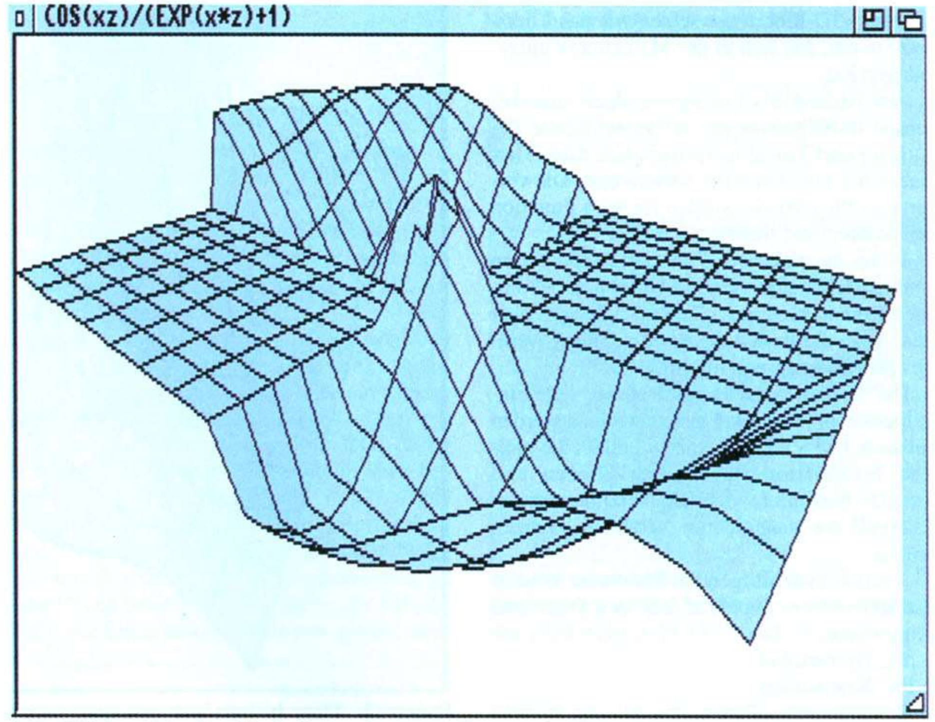
von Ilse u. Rudolf Wolf

Die mathematischen Funktionen des ARexx kann man mit der »rexxmathlib.library« von der Fish 227 ergänzen, die alle trigonometrischen Funktionen, e^x, x^y und die Fakultät (n!) liefert.

Binden Sie zusätzlich auch APIG (Fish 634) ein, können Sie in ARexx sogar Grafikprogramme schreiben, denn die Zusatzbibliotheken ermöglichen den Zugriff auf die wichtigsten Funktionen der »graphics.library« des Amiga-Betriebssystems.

Eine Anwendung zeigt der nachfolgend vorgestellte 3-D-Funktionsplotter (Listing unten). Auf die ausführliche Erklärung der mathematischen Grundlagen gehen wir hierbei nicht ein. Wer tiefer in dieses Gebiet eindringen möchte, sei auf die im Literaturverzeichnis am Ende dieses Artikels genannten Fachbücher und Artikel verwiesen.

Um räumliche Objekte am Bildschirm darzustellen, müssen Beziehungen zwischen den räumlichen Koordinaten x, y, z eines Punkts und den ebenen Bildschirm-Koordinaten u, v hergestellt werden. Zur Lösung dieses Problems wurden zahlreiche Projektionsarten entwickelt. Für die Darstellung von Funktionsgrafen am Bildschirm wird vorwiegend die Zentralprojektion angewendet. In der darstellenden Geometrie paßt sich der Beobachter dem Objekt an. In unserem Programmbeispiel ist jedoch das Projektionszentrum fix und das



3-D-Effekt: Die Grafik haben wir per ARexx mit Unterstützung der »rexxmathlib.library« programmiert – dargestellt wird die Funktion cos(xz(/(exp(x*z)+1)

```
/* ========== 3D-Plot.rexx ========== */
/* Externe Bibliotheken einbinden */
if(~show('l','rexxsupport.library'))
  then call addlib('rexxsupport.library',0,-30,0)
if(~show('l','apig.library'))
  then call addlib('apig.library',0,-30,0)
if(~show('l','rexxmathlib.library'))
  then call addlib('rexxmathlib.library',0,-30,0)

/* Intuition-Konstanten global setzen */
call set_apig_globals()
PI =3.141593
/* Messageport öffnen */
portname ="msgport"
call openport(portname)
WaitForPort portname

call default()       /* Voreinstellungen */
call parameter1()    /* Voreinstellungen ändern */
call rotation()      /* Drehung im Raum */

/* Fenster öffnen */
winidcmp=CLOSEWINDOW
flags=WINDOWCLOSE+WINDOWDRAG+WINDOWSIZING+,
      WINDOWDEPTH+GIMMEZEROZERO+ACTIVATE
win=openwindow(portname,0,0,wx,wy,2,0,winidcmp,
     ,flags,title,null(),0,0,0)
rastp=getwindowrastport(win)
call setapen(rastp,3) /* AREA-Füllfarbe */
call setopen(rastp,1) /* Outlinepen */
call plot()

do forever
   x=waitpkt(portname)
   do forever
      msg='0000 0000'x
      msg=getpkt(portname)
      if msg='0000 0000'x then leave
      class=getarg(msg,0)
      if class=CLOSEWINDOW then exitme=1
      x=reply(msg,0)
   end
 if exitme=1 then leave
end

/* Aufräumen */
a=closewindow(win)
exit

/* Zeichenroutine */
plot:
zeile1=0;zeile2=1
incz=(ze-za)/(zlines-1)
incx=(xe-xa)/(dichte-1)
netz=dichte/xlines
do z=ze to za by -incz
```

»3D-Plot.rexx«: Mit diesem Programm können Sie Funktionen per ARexx dreidimensional darstellen (Anfang)

Objekt wird durch die geeignete Wahl des Kipp- und Drehwinkels solange gedreht und gewendet, bis es sich von der gewünschten Ansicht zeigt.

Im dreidimensionalen Koordinatensystem gibt es drei Achsen: x , y und z. Hier gibt es zwei Möglichkeiten: Die z-Achse ragt aus der Zeichenebene heraus (Rechtssystem) oder in sie hinein (Linkssystem). Im Beispielprogramm »3D-Plot.rexx« setzen wir das Linkssystem ein, das sich in der Mathematik eingebürgert hat.

Am Anfang des Listings erfolgen zunächst einige Initialisierungen, die verschiedene Parameter und Transformationen betreffen. Hier haben Sie Spielraum für Änderungen. Die voreingestellten Werte wurden für jede Funktion ausprobiert und liefern ein schönes Bild.

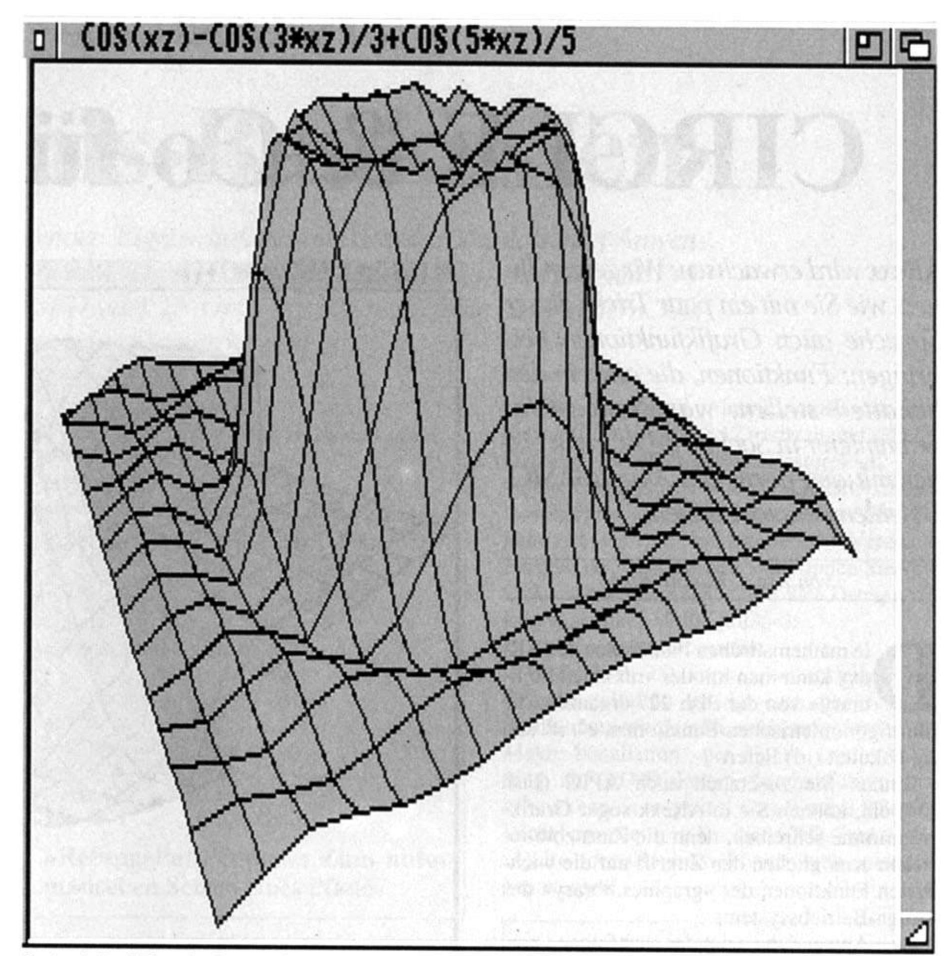
Wenn Sie eine neue Funktion ausprobieren und zunächst einen groben Überblick bekommen wollen, verringern Sie die Dichte des Netzes, da so die Erstellung der Zeichnung weniger Zeit in Anspruch nimmt.

Die Variable »dichte« gibt an, wie viele Einzelpunkte in jeder y/z-Ebene berechnet werden müssen. Die Variable »znetz« enthält die gleiche Information für die x/y-Ebenen und »xnetz« bestimmt, wie viele Netzlinien pro x-Intervall die waagrechten Netzlinien kreuzen sollen.

Auch die nachfolgenden Parameter können Sie ändern bzw. anpassen, z.B. den Dreh- und Kippwinkel:

- dw Drehwinkel
- kw Kippwinkel

Beispielsweise liefert »kw=0« die Ansicht von vorne. Positive Werte liefern einen Blick



Beispiel: Hier haben wir mit unserem Funktionsplotter die Funktion cos(xz)-cos(3*xz)/3+cos(5*xz)/5 umgesetzt (Formel 3 im Listing)

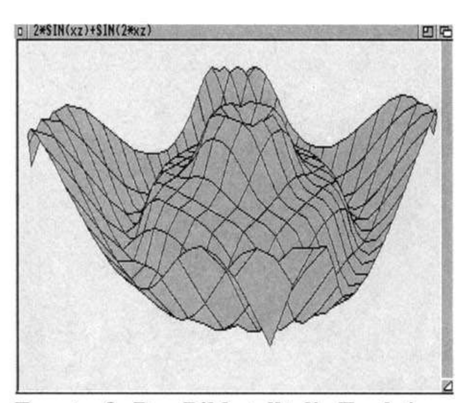
```
swap=zeile1
zeile1=zeile2
zeile2=swap
zaehler=-1; stem1=0
do x=xa to xe by incx
 /* Funktion aufrufen */
  call funktion1()
  /* Transformation */
  xtr1=sx*x;ytr1=sy*y;ztr1=sz*z
  xtr2=xtr1*w1+ztr1*w2
  ytr2=xtr1*w3+ytr1*w4+ztr1*w5
  ztr2=xtr1*w6+ytr1*w7+ztr1*w8
  u =tx-pz*(xtr2-0)/(ztr2-pz)
  v =ty+(pz*ytr2/(ztr2-pz))/2
  knoten=0
  /* Zeile speichern */
  storeX.stem1.zeile1=u
  storeZ.stem1.zeile1=v
  zaehler =zaehler+1

  /* Polygon für Netzfläche berechnen  */
  if zaehler=netz & z ~=ze & y>=ya & y<=ye then do
    zaehler=0
    do idx=stem1-netz TO stem1
      u=storeX.idx.zeile1
      v=storeZ.idx.zeile1
      u.knoten=u
      v.knoten=v
      knoten=knoten+1
    end
    do idx =stem1 TO stem1-netz by -1
      u=storeX.idx.zeile2
      v=storeZ.idx.zeile2
      u.knoten=u
      v.knoten=v
      knoten=knoten+1
    end
      /* Netzfläche zeichnen */
      a=makearea(win,640,256,100)
      a=areamove(rastp,u.0,v.0)
      do idx=1 to knoten-1
        a=areadraw(rastp,u.idx,v.idx)
      end
      a=areadraw(rastp,u.0,v.0)
      a=areaend(rastp)
      a=freearea(win)
    end
    stem1=stem1+1
    end
end
return

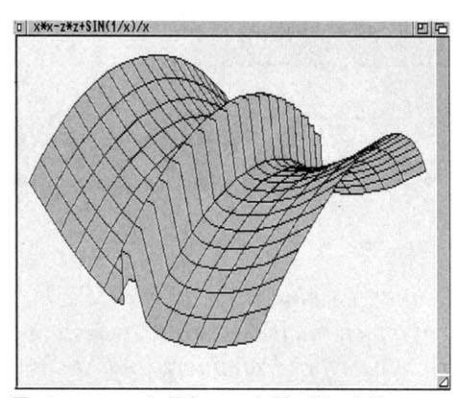
 /* Voreingestellte Parameter */
rotation:
dw=dw*PI/180 /* Drehwinkel */
kw=kw*PI/180 /* Kippwinkel */
w1=COS(kw)
w2=-SIN(kw)
w3=SIN(dw)*SIN(kw)
w4=COS(dw)
w5=SIN(dw)*COS(kw)
w6=COS(dw)*SIN(kw)
w7=SIN(dw)
w8=COS(dw)*COS(kw)
return

default:
 wx=640;wy=256      /* Fenstergröße */
 dw=30;kw=-40       /* Rotationswinkel */
 tx=wx/2; ty=wy/2   /* Ebenentranslation */
 pz=-500            /* Projektionszentrum */
 sx=3;sy=3;sz=3     /* Skalierung */
 /* Intervalle Anfang/Ende */
 xa=-63;xe=63;ya=-63;ye=63;za=-63;ze=63
```

»3D-Plot.rexx«: Mit diesem Programm können Sie Funktionen per ARexx dreidimensional darstellen (Fortsetzung)



Erraten?: Das Bild stellt die Funktion 2*sin(xz)+sin(2*xz) dar (Formel 8)



Extravagant: Diesmal die Funktion x*x-z*z+ sin(1/x)/x (Formel 2)

von oben. Negative Werte liefern eine Ansicht von unten. Voreingestellt sind »dw=-7.5« und »kw=40«.

– Die Entfernung und Lage des Projektionszentrums (Fluchtpunkt) bestimmt die Variable pz. Voreingestellt ist »pz=-500«.

– Die Skalierung der Abbildung erfolgt mit den Skalierungsfaktoren sx, sy und sz. Voreingestellt ist 3.

– Anfang und Ende des zu zeichnenden Intervalls (63 entspricht etwa 2*PI) werden durch die Raumkoordinaten xa, xe, ya, ye, za und ze festgelegt.

– Die Auflösung der Zeichnung hängt von der Größe der Netzflächen und der Anzahl der zu berechnenden Stützpunkte ab. Die Anzahl der Netzlinien pro z-Intervall wird mit »netz« bestimmt, für das x-Intervall ist »xnetz« zuständig. Die Dichte der Stützpunkte wird mit der Variablen »dichte« festgelegt.

Bevor mit dem Zeichnen des Funktionsgrafen begonnen werden kann, muß natürlich ein Fenster geöffnet werden, denn ohne ein solches läuft bekanntlich am Amiga nichts.

In den ineinander geschachtelten DO...END-Schleifen werden dann die Funktionswerte berechnet, per Zentralprojektion in die Bildschirm-Koordinaten u und v projiziert. Diese bilden die Netzknoten für die jeweilige Netzfläche und werden in ein Array geschrieben. Ist eine Fläche berechnet, wird sie mit der AREAFILL-Funktion gefüllt. Weil von hinten nach vorne gezeichnet wird, ist auch das Problem der verdeckten Linien und Flächen gelöst, weil alles was dahinter gezeichnet wurde, mit der AREA-Funktion übermalt wird.

An das eigentliche Programm wurden Unterprogramme mit den Definitionen für acht Funktionen und den dazu geeigneten Parametern gehängt. Die zu zeichnende wird mit »call parameterX()« in Zeile 20 und mit »call funktionX()« in Zeile 62 ausgewählt. ■

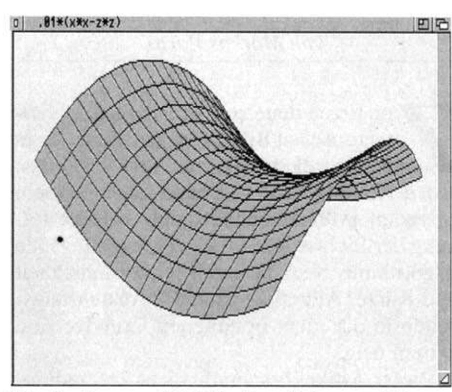
Literatur:

[1] Walter Wunderlich: Darstellende Geometrie I u. II, BI-Hochschultaschenbuch 1966/67

[2] R.E. Myer: Microcomputer Graphics, Addison Wesley 1982

[3] Markus Weber: 3-D-Grafik, IWT-Verlag 1984

[4] Dr.G.Glaeser: 3-D-Programmierung mit BASIC, Mikrocomputer-Praxis - B.G.Teubner 1986



Einfach: y = 0,01*(x*x-z*z) per ARexx ins Bild gebracht (Formel 1)

```
 /* Größe einer Netzfläche */
 zlines=20 /* Linien pro z-Intervall */
 xlines=15 /* Linien pro x-intervall */
 faktor=2;dichte=faktor*xlines
return

/* **** voreingestellte Funktionen     **** */
/* **** siehe auch Bilder im Artikel **** */
funktion1:
 y=(x*x-z*z)/100
return
parameter1:
 pz=-1000;za=-52
 dw=30;kw=-30
 title=".01*(x*x-z*z) (Sattelfläche)"
return

funktion2:
 y=.01*(x*x-z*z)+140*(SIN(1/x)/x)
return
parameter2:
  ty=wy/3+30;pz=-600
  title="x*x-z*z+SIN(1/x)/x"
return

funktion3:
 xz=SQRT(x*x+z*z)/10
 y=20*(COS(xz)-COS(3*xz)/3+COS(5*xz)/5)
return
parameter3:
  ty=80;dw=30;kw=-60
  sx=5.5;sy=5.5;sz=5.5
  xa=-30;xe=30;ya=-30;ye=30;za=-30;ze=30
  title="COS(xz)-COS(3*xz)/3+COS(5*xz)/5"
return

funktion4:
  xz=SQRT(x*x+z*z)/10
  y=20*SIN(xz)
return
parameter4:
  ty=wy/3
  dw=30;kw=-40
  title="SIN(xz)"
return

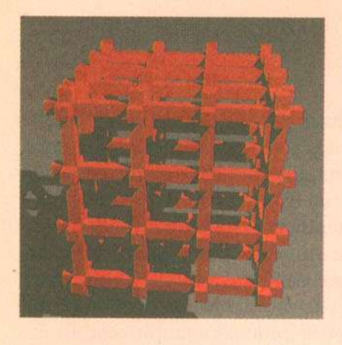
funktion5:
  xz=SQRT(x*x+z*z)/10
  y=20*COS(xz)/(EXP(x*z)+1)
return
parameter5:
  tx=wx/2-20;ty=wy/2-20
  sx=4;sy=4;sz=4
  za=-40;ze=50
title="COS(xz)/(EXP(x*z)+1)"
return

funktion6:
 y =SQRT(x*x/4+z*z/9)
return
parameter6:
  ty=wy/2+20; pz=-600
title="SQRT(x*x/4+z*z/9)"
return

funktion7:
 xz=SQRT(x*x+z*z)/10
 y=-30*COS(xz)*EXP(-xz/15)
parameter7:
 ty=wy/3
 title="COS(xz)*EXP(-xz/15)"
return

funktion8:
 xz=SQRT(x*x+z*z)/10
 y=10*(2*SIN(xz)+SIN(2*xz))
return
parameter8:
 ty=wy/3
 title="2*SIN(xz)+SIN(2*xz)"
return
```


»3D-Plot.rexx«: 3-D-Plotter für Funktionen in ARexx – acht Formeln bzw. Bilder sind bereits programmiert



Octree: Die fantas

3-D-Grafiken mit dem Computer zu berechnen ist eine faszinierende Sache. Besonders, wenn man realistische Bilder berechnet, nach dem sog. Raytracing-Verfahren. Wir zeigen Ihnen, wie man so etwas programmiert und stellen Ihnen sogar eine Verbesserung – sprich Beschleunigung – des gängigen Octree-Algorithmus und seine Implementierung vor.

von Markus Porto

Zur Erzeugung von künstlichen, photorealistischen Bildern bedient man sich in der Regel des Raytracing-Verfahrens, also der Ermittlung der Lichtintensität an einem bestimmten Rasterpunkt durch Rückverfolgung der Lichtstrahlen. Der zugrundeliegende Algorithmus besticht durch seine Einfachheit und Kürze. Allerdings sind die Ausführungszeiten in der nicht optimierten Grundversion extrem lang.

Dieser Artikel beschreibt nicht das traditionelle Verfahren zur Bildberechnung mit Hilfe des Raytracing-Verfahrens – das wird detailliert und ausführlich in [1] vorgestellt, dem Standardwerk zur Computergrafik schlechthin, aber auch in [2] oder [3]. Hier geht's um eine Erweiterung des Algorithmus´, der in seiner Grundform sehr einfach, leider aber auch sehr langsam ist.

Gut Ding will Weile haben

Wir stellen die Probleme des traditionellen Verfahrens vor, zeigen Techniken zur Beschleunigung, gehen auf den Octree-Algorithmus ein, widmen uns dem Strahlengenerator von Müller sowie Verfahren von Glassner, Samet und Endl zur Suche einer Nachbarzelle.

Die Ursache für den hohen Zeitaufwand, den das Raytracing-Verfahren in der Grundversion verursacht, ist die Tatsache, daß für jeden Rasterpunkt die Schnittpunkte des Sehstrahls mit allen Objekten bestimmt werden. Zwei Strategien zur Beschleunigung setzen genau hier an:

❐ Es wird versucht, die Teile des Programms zu optimieren, in denen diese Schnittpunkte berechnet werden. Zum einen erreicht man dies durch geschickte Wahl der Parametrisierung der Objekte (z.B. eine Kugel wirklich als Kugel und nicht durch eine Vielzahl Dreiecke anzunähern), zum anderen dadurch, daß komplexe Objekte mit einem umschließendes Volumen umhüllt werden (im einfachsten Fall ein »Voxel«, ein Quader, dessen Seitenflächen zu den Raumachsen parallel sind), so daß sich Schnittpunkte relativ einfach aufspüren lassen; und nur wenn ein solcher Schnittpunkt des Sehstrahls mit diesem Volumen vorliegt, schneidet man das eigentliche Objekt.

❐ Eine zweite Möglichkeit, die Strahlenverfolgung zu optimieren: Man versucht, nur die Objekte zu schneiden, für die überhaupt ein Schnittpunkt möglich ist, die also »in der Nähe« des Sehstrahls liegen.

Vergleicht man die zweite mit der ersten Variante, stellt man schnell fest, daß letztere Strategie erfolgversprechender ist, da bei der ersten weiterhin für alle Objekte mindestens ein Schnittpunkt (und sei es der mit dem umschließenden Volumen) gefunden werden muß.

Es ist allerdings nicht einfach, festzustellen, welche Objekte in unmittelbarer Nähe des Sehstrahls liegen. Zu diesem Zweck teilt man den von den Objekten eingenommenen Raum in kleinere Einheiten auf und ordnet jeder die Menge an Objekten zu, deren Oberfläche ganz oder teilweise darin liegen. »Schickt« man dann den Sehstrahl durch den Raum, ermittelt man die Einheiten, die auf seinem Weg liegen und schneidet nur die Objekte, die diesen Untereinheiten zugeordnet sind.

Auch hier kennt man zwei Strategien, die helfen, den Raum zu unterteilen. Beide haben Vor- und Nachteile:

❐ Zum einen läßt sich der Raum regelmäßig aufteilen. Das hat den Vorteil, daß die Verwaltung und Datenstruktur simpel zu realisieren sind und das Problem der Nachbarsuche durch direkte Adressierung zu lösen ist. Der Nachteil: Der Speicherbedarf ist sehr hoch (n^3 Zellen sind zu verwalten, wenn n die Anzahl der Unterteilungen pro Achse ist), auch die Kodierung der Objekte für jede Zelle muß neu vorgenommen werden, was zeitaufwendig ist. Hinzu kommt, daß bei feiner Raumaufteilung sehr viele Nachbarn gefunden werden müssen.

Devise: Teile und herrsche

❐ Zum anderen ist es möglich, jede Zelle rekursiv in acht Kinderzellen (sog. Oktanten, daher der Name »Octree«) zu unterteilen. Dabei wird die betreffende Zelle jeweils in der Mitte durch drei zu den Seitenflächen parallele Ebenen zerschnitten, so daß ein linker und rechter, ein unterer und oberer sowie ein hinterer und vorderer Bereich gebildet werden. Damit entstehen acht Unterzellen, von links-unten-hinten bis rechts-oben-vorne. Das wird solange durchgeführt, bis eine maximale Tiefe erreicht oder aber in der zu unterteilenden Zelle nur noch ein oder gar kein Objekt mehr enthalten ist. Der Vorteil: Es erfolgt eine automatische Anpassung an die gegebene Szene und Teile des Raums, die wenige oder keine Objekte enthalten, werden kaum oder gar nicht unterteilt. Gleichzeitig verringert sich so der Speicherbedarf und ermöglicht eine höhere Auflösung. Dabei enthalten nur die Zellen, die ein Blatt dieses Baums darstellen, eine Objektliste, alle anderen verweisen auf ihre Kinderzellen.

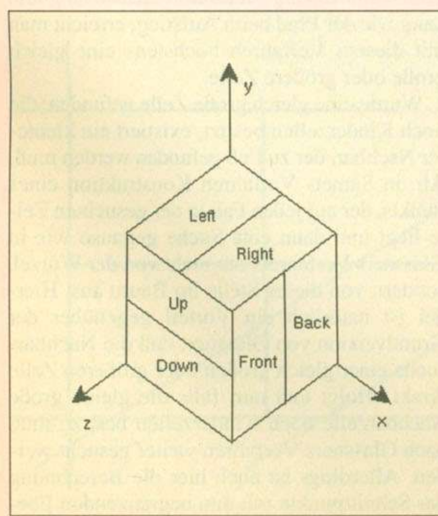
Der Aufbau läßt eine starke Abhängigkeit zwischen den Zellen entstehen, da z.B. Objekte, die in einer Zelle nicht mehr enthalten sind, auch nicht in den Kinderzellen vorkommen können. Außerdem müssen weniger Nachbarzellen für einen Strahlendurchlauf berechnet

Beispiel: Das ist Raytracing – mehrere Obj solche Objekte zu berechnen braucht man s

ytracing-Verfahren

ische Welt des Computers

, die aus unterschiedlichen geometrischen Einheiten zusammengesetzt sind. Um elle Rechner und Algorithmen.



1. Skizze: Unterteilung des Raums

werden. Die Nachteile: Der Aufbau und die Verwaltung der Baumstruktur (jeder Knoten hat acht Kinderknoten) ist aufwendiger und schwieriger als bei einer gleichmäßigen Unterteilung. Auch ist die Nachbarsuche wesentlich komplizierter, vor allem dann, wenn der Baum nicht jedesmal von der Wurzel aus durchlaufen werden soll.

In diesem Artikel stellen wir eine Technik vor, welche die Vorteile der vorgestellten Strategien kombiniert und die angesprochenen Nachteile reduziert. Sie werden kein komplettes Raytracing-Programm vorfinden, sondern nur die Routinen, die für dieses Verfahren nötig sind, mit einigen Tips, wie man sie am günstigsten in ein bestehendes Programm einfügt. Das hat vor allem den Grund, da ein einfacher Raytracer mit einigen wenigen Grundobjekten zwar sehr schnell geschrieben ist, trotzdem äußerst umfangreich ist. Zudem sind die dabei

Was ist Raytracing?

Oder: Was Sie schon immer über Raytracing wissen wollten

Durch das Raytracing-Verfahren wird versucht, eine möglichst fotorealistische Darstellung von Objekten mit Hilfe eines Computers zu erhalten.

Prinzipiell müßte man dafür die Bahn der von den vorhandenen Lichtquellen ausgehenden Lichtstrahlen untersuchen. Allerdings würde dieses Verfahren ungeheuer lange dauern, da nur ein minimaler Teil der ausgesendeten Lichtstrahlen je den Beobachter erreicht. Also nutzt man geschickt die Umkehrbarkeit des Lichtwegs aus.

Das Prinzip, nach dem ein Raytracer arbeitet, ist, vom Beobachter aus durch jedes Pixel einen Sehstrahl in die Szene zu schicken und nach Schnittpunkten mit den Objekten zu suchen. Hat man einen Schnittpunkt gefunden, bleibt zu entscheiden, welche Farbe das Objekt an der betreffenden Stelle hat, ob es durchlässig oder verspiegelt ist. Treffen die beiden letzten Eigenschaften zu, wird vom Schnittpunkt aus je ein neuer Strahl gestartet, bis dieser wieder ein Objekt schneidet usw.. Für die Ermittlung der Farbe am Schnittpunkt existieren verschiedene Beleuchtungsmodelle. Da das Verhalten von Licht an Oberflächen sehr kompliziert ist, werden verschiedene Vereinfachungen getroffen und einige Feinheiten vernachlässigt. Ein einfaches Beispiel für ein solches Beleuchtungsmodell ist das von Phong entwickelte, welches sehr grobe Annäherungen an die Wirklichkeit macht, die aber für Plastik erstaunlich gut zutreffen (daher sehen auch die Objekte in mit diesem Beleuchtungsmodell berechneten Bildern alle mehr oder minder nach Plastik aus).

Damit ist es allerdings noch nicht getan, denn die oben angeschnittenen Techniken lassen die Bilder nicht besonders realistisch aussehen, da sie zu perfekt wirken. Kein Objekt in der Wirklichkeit ist gleichmäßig durchlässig oder verspiegelt, ist von gleichmäßiger Farbe oder besitzt solch scharfe Kanten oder Ecken. Daher gilt es dann z.B. noch, sog. Texturen zu definieren, die entweder die Farbe oder den Normalen-Vektor vom Ort abhängig so ändern, daß die Oberfläche z.B. wie Marmor oder Wasser aussieht. Oder an Grenzflächen zwischen Objekten überzublenden, so daß scharfe Kanten vermieden werden, usw.

Allerdings stößt man selbst bei wenigen Objekten in einer Szene oft schon an die Grenzen der Leistungsfähigkeit des zur Verfügung stehenden Rechners. Rechenzeiten für ein Bild von Stunden oder Tagen sind bei aufwendigen Szenen keine Seltenheit. Das hier beschriebene Verfahren erlaubt es, fast unabhängig von der Anzahl der Objekte Bilder in fast konstanter Zeit zu berechnen.

verwendeten Verfahren und Techniken – wie eingangs erwähnt – in der angegebenen Fachliteratur ausführlich erläutert.

Zum Verständnis dieses Verfahrens ist es aber nötig, die grundlegenden Algorithmen der oben angesprochenen Strategien zur Unterteilung des Raumes darzustellen, um sie anschließend miteinander zu verknüpfen.

Strahlengenerator: Beam me up, Scotty

Da ist zum einen der »Strahlengenerator« von Müller (siehe Skizze rechts): Er verallgemeinert des Verfahren zur Generierung einer Geraden auf einem Rasterschirm. Voraussetzung ist dabei eine äquidistante Unterteilung eines Quaders, der die gesamte Szene enthüllt in »xCount x yCount x zCount« Zellen.

Die Abfolge der Zellen, die der Strahl durchläuft, wird dabei inkrementell bestimmt. Dabei wird ausgenutzt, daß alle die Zellen begrenzenden Ebenen parallel liegen und die gleichen Abstände »xLength«, »yLength« und »zLength« besitzen. Sei nun »start« der Anfangspunkt des Sehstrahls in der Zelle mit den Indizes »(xIndex, yIndex, zIndex)« und »ray« der normierte Richtungsvektor. Aus den Strahlensätzen folgt für die Abstände der Ebenen entlang des Richtungsvektors

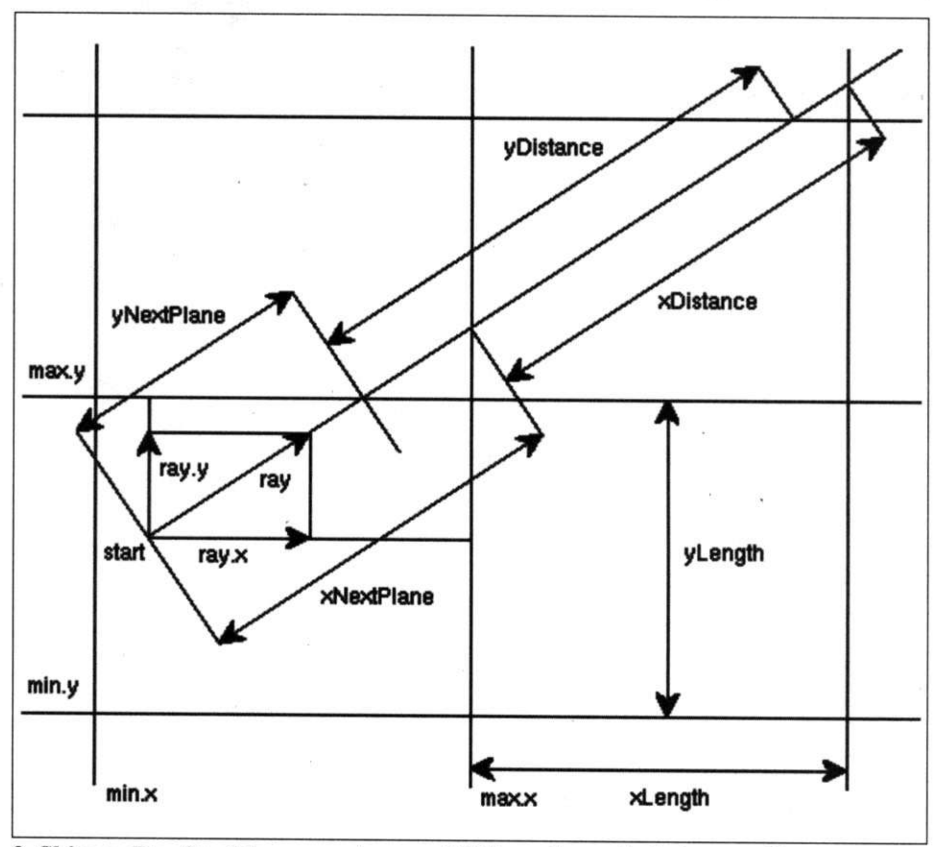
```
xDistance = Abs( xLength/ray.x)
yDistance = Abs( yLength/ray.y)
zDistance = Abs( zLength/ray.z)
```

Falls eine der Komponenten von »ray« Null bzw. betraglich kleiner als eine definierte Zahl »Epsilon« ist, wird die Größe, die i.a. per Division durch diese Komponente bestimmt wird, einfach auf eine sehr große Zahl (im Programm das Makro »INFINITY«) gesetzt und vom Algorithmus als unendlich behandelt. Für den Abstand der nächsten Ebene in X-Richtung entlang des Richtungsvektors (gilt genauso für die Y- und Z-Koordinaten) vom Startpunkt aus, ergibt sich ebenfalls nach dem Strahlensatz:

```
xNextPlane = (min.x-start.x)/ray.x
```
und
```
xDirection =- 1
```
falls
```
ray.x < 0
```
und
```
xNextPlane = (start.x-max.)/ray.x
```
und andernfalls
```
xDirection = 1
```
wobei »min« und »max« die minimale und maximale Ausdehnungen der Zelle sind, in der sich der Startpunkt befindet. Dann ist immer die Ebene die nächste geschnittene, zu der der Abstand entlang des Richtungsvektors minimal ist. Ist so eine Ebene gefunden, ergeben sich die neuen Abstände so:

Die beiden anderen Abstände werden um diesen minimalen Abstand verringert, da man um genau dieses Maß entlang des Richtungsvektors gelaufen ist. Der verbleibende dritte Abstand wird auf den Abstand der Ebenen entlang des Richtungsvektors in dieser Richtung gesetzt, da der neue Ausgangspunkt sich ja auf einer der Ebenen befindet (Listing rechts oben). Nur am Anfang wird die Position der Zelle und des Startpunkts benutzt, alles weitere wird inkrementell durchgeführt.

Des weiteren wird noch ein Verfahren benötigt, das von einer Zelle ausgehend den Nachbarn in einer bestimmten Richtung findet. Das läßt sich zum einen durch Glassners Verfahren erreichen, wobei man die Schnittpunkte der durch den Sehstrahl gebildeten Geraden mit den begrenzenden Ebenen berechnet, den am nähesten gelegenen ermittelt, daraus einen Punkt konstruiert, der auf jedem Fall in der Nachbarzelle liegt, und dann von der Wurzel ausgehend die kleinste Zelle sucht, die diesen Punkt enthält. Der Nachteil: Es müssen Schnittpunkte mit den begrenzenden Ebenen und ein »sicherer« Punkt berechnet werden, außerdem ist der Baum jedesmal von der Wurzel aus zu durchlaufen.



2. Skizze: Der Strahlengenerator von Müller schematisch dargestellt

Bäumchen wechsle dich

Zum anderen existiert noch eine Weiterentwicklung des Verfahrens nach Samet. Die Aufgabe dabei ist, zu einer gegebenen Zelle und einer gegebenen Richtung einen gleich großen oder größeren Nachbarn zu finden. Man steigt dabei von der Startzelle ausgehend im Baum auf, bis der tiefste gemeinsame Vorfahre der Ausgangszelle und der gesuchten Nachbarzelle gefunden wurde. Dann steigt man den an der Schnittebene gespiegelten Pfad wieder ab, bis entweder die gleiche Tiefe erreicht ist oder die Zelle keine Kinderzellen mehr hat. Dabei ist der tiefste gemeinsame Vorfahre in einer bestimmten Richtung der, der zuerst über einen Sohn aus der anderen Richtung erreicht wird. Falls man also nach links gehen möchte, muß man den ersten Vorfahren finden, zu dessen rechten Teilbäumen die Ausgangszelle gehört.

Das Spiegeln des Pfads erreicht man, indem man immer die Richtung im Pfad herumdreht, in die man geht. Falls man also nach links geht, muß man beim Absteigen nach rechts gehen (aus einem linken, oberen, hinteren wird ein rechtes, oberes, hinteres Kind usw.). Da der Pfad beim Abstieg höchstens genauso lang sein kann wie der Pfad beim Aufstieg, erreicht man mit diesem Verfahren höchstens eine gleich große oder größere Zelle.

Wurde eine gleich große Zelle gefunden, die noch Kinderzellen besitzt, existiert ein kleinerer Nachbar, der zu Fuß gefunden werden muß, d.h. in Samets Verfahren Konstruktion eines Punkts, der auf jeden Fall in der gesuchten Zelle liegt und dann eine Suche genauso wie in Glassners Verfahren, nur nicht von der Wurzel, sondern von dieser Stelle im Baum aus. Hierbei ist natürlich ein Vorteil gegenüber der Grundversion von Glassner, daß die Nachbarsuche einer gleich großen oder größeren Zelle direkt erfolgt und nur, falls die gleich große Nachbarzelle noch Kinderzellen besitzt, muß nach Glassners Verfahren weiter gesucht werden. Allerdings ist auch hier die Berechnung der Schnittpunkte mit den begrenzenden Ebenen sowie eines sicheren Punkts nötig.

```
WHILE (xIndex,yIndex,zIndex) > (1, 1, 1) AND
      (xIndex,yIndex,zIndex) < (xCount,yCount,zCount) DO
 BEGIN
  IF (xNextPlane < yNextPlane) AND (xNextPlane < zNextPlane) THEN

    BEGIN    /* Strahl schneidet als nächstes eine yz-Ebene */
    INC( xIndex, xDirection);
    yNextPlane -= xNextPlane;
    zNextPlane -= xNextPlane;
    xNextPlane= xDistance;

  END

  ELSE
    IF (yNextPlane < zNextPlane) THEN

    BEGIN       /* Strahl schneidet als nächstes eine xz-Ebene */
      INC( yIndex, yDirection);
      xNextPlane -= yNextPlane;
      zNextPlane -= yNextPlane;
      yNextPlane= yDistance;

    END

    ELSE

    BEGIN       /* Strahl schneidet als nächstes eine xy-Ebene */
     INC( zIndex, zDirection);
      xNextPlane -= zNextPlane;
      yNextPlane -= zNextPlane;
      zNextPlane = zDistance;

   END; /* of if */
  END; /* of if */
END; /* of while */
```

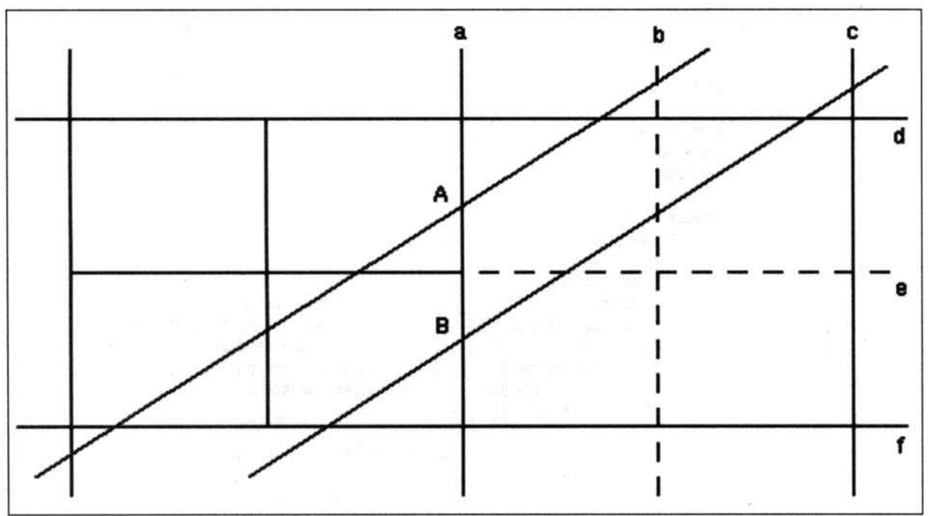

»Grundlegend«: Der Algorithmus von Müller in Form eines Pascal-Listings. Für jeden Strahl wird geprüft, ob er das Objekt schneidet.

So lasset die Spiele beginnen

Kommen wir zur Umsetzung unseres verbesserten Octree-Algorithmus´ in ein Programm. Auf den nächsten Seiten finden Sie den wichtigsten Teil »Octree.c«. Es handelt sich um das Hauptmodul. Auf den weiteren Seiten zeigen wir Ihnen die zusätzlich erforderlichen Include- und Definitionsdateien.

Das Ziel war es, ein Verfahren zu finden, das die Vorteile von Müllers Verfahren mit denen von Samets verbindet, das also die Bestimmung der Richtung der Nachbarzelle inkrementell erlaubt und diese Zelle nur durch Auf- und Absteigen im Octree findet, ohne daß dafür ein sicherer Punkt in der Nachbar-Zelle nötig ist. Die entscheidende Idee hatte dabei Dipl. Math. Robert Endl an der Universität Marburg!

Als Voraussetzungen für dieses Verfahren sind nur die Kenntnis der Tiefe der momentanen Zelle im Baum, die für jede Tiefe konstanten Größen »xDistance, »yDistance«, »zDistance« sowie die Abstände »xNextPlane«, »yNextPlane«, »zNextPlane« nötig (Skizze unten). Aus den Größen »xDirection«, »yDirection« und »zDirection« in Müllers Verfahren wird hier dann die Hauptrichtung des Sehstrahls (im Programm »mainDirection«), die angibt, ob der Sehstrahl nach links oder rechts, nach unten oder oben, nach hinten oder vorne läuft. Die Möglichkeiten gehen dann wie bei der Richtung der Kinder einer Zelle von links – unten – hinten bis rechts – oben – vorne. Falls im weiteren von der Richtung des Sehstrahls die Rede ist, ist immer diese Hauptrichtung bzw. eine ihrer Komponenten gemeint.

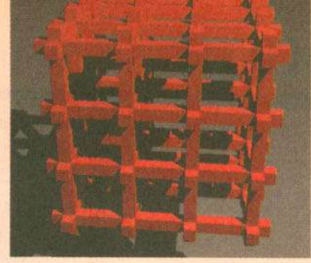


3. Skizze: Adaption der Größen nach dem Verfahren von Robert Endl

Auf der Jagd nach Schnittpunkten

Nachdem man den Sehstrahl mit den der aktuellen Zelle zugeordneten Objekten geschnitten und keinen endgültigen Schnittpunkt gefunden hat (ein Schnittpunkt ist dann endgültig, falls er in der gerade bearbeiteten Zelle liegt; man ist in einem solchen Fall fertig, da keine näheren mehr gefunden werden können), wird die nächste zu behandelnde Zelle gesucht. Als erstes wird mit Hilfe des Strahlengenerators von Müller die Richtung des nächsten Nachbarn ermittelt. Dann wird in dieser Richtung mit Samets Verfahren eine gleich große oder größere Nachbar-Zelle gefunden. Dabei werden beim Auf- wie auch beim Absteigen sowohl die Größen »xDistance«, »yDistance« und »zDistance« als auch »xNextPlane, »yNextPlane« und »zNextPlane« der größeren bzw. kleineren Zelle angepaßt.

Die ersteren können einer Tabelle entnommen werden, die am Anfang eines Laufs be-



```
/* File octree.c */

#include "typedefs.h"
#include "import.h"
#include "mathdefs.h"
#include "intersection.h"
#include "voxel.h"
#include "object.h"
#include "export.h"
#include "octree.h"

#define MAXOCTREEDEPTH  6
typedef struct tcontent        /* content */
{struct tcontent *next;
 OBJECT   *object;
} CONTENT;
typedef struct toctree         /* octree */
{struct toctree *parent;
 WORD   direction;
 VOXEL   voxel;
 CONTENT   *content;
 BOOLEAN  hasChildren;
 struct toctree *child[8];
} OCTREE;

#define BACK            0x00
#define FRONT           0x01
#define DOWN            0x00
#define UP              0x02
#define LEFT            0x00
#define RIGHT           0x04
#define BACKFRONT       0x01
#define DOWNUP          0x02
#define LEFTRIGHT       0x04
#define NODIRECTION     -1
#define LEFTDOWNBACK    (LEFT|DOWN|BACK)
#define LEFTDOWNFRONT   (LEFT|DOWN|FRONT)
#define LEFTUPBACK      (LEFT|UP|BACK)
#define LEFTUPFRONT     (LEFT|UP|FRONT)
#define RIGHTDOWNBACK   (RIGHT|DOWN|BACK)
#define RIGHTDOWNFRONT  (RIGHT|DOWN|FRONT)
#define RIGHTUPBACK     (RIGHT|UP|BACK)
#define RIGHTUPFRONT    (RIGHT|UP|FRONT)
#define ISBACK(x)       (((x) & BACKFRONT) == BACK)
#define ISFRONT(x)      (((x) & BACKFRONT) == FRONT)
#define ISDOWN(x)       (((x) & DOWNUP)    == DOWN)
#define ISUP(x)         (((x) & DOWNUP)    == UP)
#define ISLEFT(x)       (((x) & LEFTRIGHT) == LEFT)
#define ISRIGHT(x)      (((x) & LEFTRIGHT) == RIGHT)
LOCAL OCTREE *octreeRoot;
LOCAL WORD maxOctreeDepth;
LOCAL ULONGWORD identification;
LOCAL FLOATING xLength[MAXOCTREEDEPTH+1];
LOCAL FLOATING yLength[MAXOCTREEDEPTH+1];
LOCAL FLOATING zLength[MAXOCTREEDEPTH+1];
LOCAL VOID MinimizeMaximizeObject( OBJECT *header,
                                   VOXEL *voxel)
{OBJECT *help;
 help= header;
 while( help != (OBJECT *)NULL)
 {if( !help->unlimited)
  {UnionVoxel( voxel, voxel, &help->voxel);
  }
  help= help->next;
 }
}
LOCAL CONTENT *AllocateContent( CONTENT *next,
                                OBJECT *object)
{CONTENT *content;
 if( (content= Allocate( CONTENT)) != (CONTENT *)NULL)
 {content->next= next;
  content->object= object;
 }
 return( content);
}
LOCAL VOID FreeContentList( CONTENT *content)
{CONTENT *help, *next;
 help= content;
 while( help != (CONTENT *)NULL)
 { next= help->next;
  Free( help);
  help= next;
 }
}
LOCAL CONTENT *AllocateContentList( OBJECT *object)
{OBJECT *help;
 CONTENT *content, *old;
 content= (CONTENT *)NULL;
 help= object;
 while( help != (OBJECT *)NULL)
 {help->identification= (LONGWORD)0;
  old= content;
  if( (content=AllocateContent(old, help)) !=
       (CONTENT *)NULL)
  {help= help->next;
  }
  else
  {FreeContentList( old);
   help= (OBJECT *)NULL;
  }
 }
 return( content);
}
LOCAL OCTREE *AllocateOctree( OCTREE *parent,
                              CONST WORD direction,
                              CONST VOXEL *voxel)
{OCTREE *octree;
 if( (octree= Allocate( OCTREE)) != (OCTREE *)NULL)
 {octree->parent= parent;
  octree->direction= direction;
  octree->voxel= *voxel;
  octree->content= (CONTENT *)NULL;
  octree->hasChildren= FALSE;
  octree->child[LEFTDOWNBACK]=
  octree->child[LEFTDOWNFRONT]=
  octree->child[LEFTUPBACK]=
  octree->child[LEFTUPFRONT]=
  octree->child[RIGHTDOWNBACK]=
  octree->child[RIGHTDOWNFRONT]=
  octree->child[RIGHTUPBACK]=
  octree->child[RIGHTUPFRONT]= (OCTREE *)NULL;
 }
 return( octree);
}
LOCAL VOID FreeOctree( OCTREE *octree)
{if( octree != (OCTREE *)NULL)
 {FreeOctree( octree->child[LEFTDOWNBACK]);
  FreeOctree( octree->child[LEFTDOWNFRONT]);
  FreeOctree( octree->child[LEFTUPBACK]);
  FreeOctree( octree->child[LEFTUPFRONT]);
  FreeOctree( octree->child[RIGHTDOWNBACK]);
  FreeOctree( octree->child[RIGHTDOWNFRONT]);
  FreeOctree( octree->child[RIGHTUPBACK]);
  FreeOctree( octree->child[RIGHTUPFRONT]);
  FreeContentList( octree->content);
  Free( octree);
 }
}
LOCAL VOID DivideOctree( OCTREE *octree,
                         CONST WORD depth)
{WORD direction;
 VECTOR mid;
 VOXEL voxel, newvoxel;
 OBJECT *object;
 CONTENT *content, *help, *old;
 if( (depth < maxOctreeDepth) &&
  ((content= octree->content) != (CONTENT *)NULL) &&
  (content->next != (CONTENT *)NULL))
 {voxel= octree->voxel;
  mid.x= 0.5*(voxel.min.x+voxel.max.x);
  mid.y= 0.5*(voxel.min.y+voxel.max.y);
  mid.z= 0.5*(voxel.min.z+voxel.max.z);
  mid.w= 1.0;
   for( direction = LEFTDOWNBACK;
        direction <= RIGHTUPFRONT; direction++)
  {
newvoxel.min.x=(ISLEFT(direction) ? voxel.min.x : mid.x);
newvoxel.min.y=(ISDOWN(direction) ? voxel.min.y : mid.y);
newvoxel.min.z=(ISBACK(direction) ? voxel.min.z : mid.z);
newvoxel.min.w= 1.0;
newvoxel.max.x=(ISLEFT(direction) ? mid.x : voxel.max.x);
newvoxel.max.y=(ISDOWN(direction) ? mid.y : voxel.max.y);
newvoxel.max.z=(ISBACK(direction) ? mid.z : voxel.max.z);
   newvoxel.max.w= 1.0;
   if( (octree->child[direction]=
    AllocateOctree( octree, direction, &newvoxel)) !=
    (OCTREE *)NULL)
   {content= (CONTENT *)NULL;
    help= octree->content;
    while( help != (CONTENT *)NULL)
    {object= help->object;
     if( (*object->VoxelIntersectedByObject)( object,
          &newvoxel))
     {old= content;
      if( (content= AllocateContent( old, object)) !=
          (CONTENT *)NULL)
      {help= help->next;
      }
      else
      {FreeContentList( old);
       help= (CONTENT *)NULL;
      }
     }
     else
     {help= help->next;
     }
    }
    octree->child[direction]->content= content;
    DivideOctree( octree->child[direction], depth+1);
   }
  }
  octree->hasChildren= TRUE;
  FreeContentList( octree->content);
  octree->content= (CONTENT *)NULL;
 }
}
GLOBAL BOOLEAN IntersectionWithOctree( OBJECT *header,
          INTERSECTION *intersection)
{WORD depth, runDepth, oldDepth;
 WORD direction, mainDirection, currentDirection,
      currentDirectionMask;
 WORD directions[MAXOCTREEDEPTH+1];
 BOOLEAN intersect, found;
 BOOLEAN xRayNotZero, yRayNotZero, zRayNotZero;
 FLOATING xNextPlane, yNextPlane, zNextPlane, lambda;
 FLOATING xInvRay, yInvRay, zInvRay;
 FLOATING xDistance[MAXOCTREEDEPTH+1];
 FLOATING yDistance[MAXOCTREEDEPTH+1];
 FLOATING zDistance[MAXOCTREEDEPTH+1];
 VECTOR start, location;
 OBJECT *help;
 CONTENT *content;
 OCTREE *octree;

 /* next ray */
 identification++;
 intersect= FALSE;

 /* unlimited objects */
 help= header;
 while( help != (OBJECT *)NULL)
 {if( help->unlimited &&
   (help->IntersectionWithObject !=
                (INTERSECTIONOBJECTPTR)NULL) &&
   (*help->IntersectionWithObject)( help, intersection))
  {location.x= intersection->start.x +
            lambda*intersection->ray.x;
   location.y= intersection->start.y +
            lambda*intersection->ray.y;
   location.z= intersection->start.z +
            lambda*intersection->ray.z;
   location.w= 1.0;
   intersect= TRUE;
  }
  help= help->next;
 }

 /* initialize */
 octree= octreeRoot;
 depth= 0;
 mainDirection= ((intersection->ray.x < 0.0) ? LEFT :
  RIGHT) | ((intersection->ray.y < 0.0) ? DOWN : UP) |
        ((intersection->ray.z < 0.0) ? BACK : FRONT);
 if( PointInVoxel( &intersection->start, &octree->voxel))
 { /* start is in root's voxel */
  start= intersection->start;
 }
 else
 {if( RayVoxelIntersection( &lambda,
   &intersection->start,
   &intersection->ray, &octree->voxel))
  {/* ray hits root's voxel */
```

```
   lambda += EPSILON;
   start.x= intersection->start.x +
           lambda*intersection->ray.x;
   start.y= intersection->start.y +
           lambda*intersection->ray.y;
   start.z= intersection->start.z +
           lambda*intersection->ray.z;
   start.w= 1.0;
  }
  else
  {/* ray miss' root's voxel */
   return( intersect);
 }}
 /* find the start voxel */
 while( octree->hasChildren)
 {currentDirection= NODIRECTION;
  direction= LEFTDOWNBACK;
  do
  {if( PointInVoxel( &start,
                    &octree->child[direction]->voxel))
   {currentDirection= direction;
   }
   direction++;
  }
  while( (currentDirection == NODIRECTION) &&
   (direction <= RIGHTUPFRONT));
  octree= octree->child[currentDirection];
  depth++;
 }
 /* precalculating */
 if( xRayNotZero= (Abs( intersection->ray.x) > EPSILON))
 {xInvRay= 1.0/intersection->ray.x;
 }
 if( yRayNotZero= (Abs( intersection->ray.y) > EPSILON))
 {yInvRay= 1.0/intersection->ray.y;
 }
 if( zRayNotZero= (Abs( intersection->ray.z) > EPSILON))
 {zInvRay= 1.0/intersection->ray.z;
 }
 for( runDepth= 0; runDepth <= maxOctreeDepth;
      runDepth++)
 {xDistance[runDepth]= (xRayNotZero ?
   Abs( xLength[runDepth]*xInvRay) : INFINITY);
  yDistance[runDepth]= (yRayNotZero ?
   Abs( yLength[runDepth]*yInvRay) : INFINITY);
  zDistance[runDepth]= (zRayNotZero ?
   Abs( zLength[runDepth]*zInvRay) : INFINITY);
 }
 if( intersection->ray.x < 0.0)
 {xNextPlane= (xRayNotZero ?
   (octree->voxel.min.x-start.x)*xInvRay : INFINITY);
 }
 else
 {xNextPlane= (xRayNotZero ?
   (octree->voxel.max.x-start.x)*xInvRay : INFINITY);
 }
 if( intersection->ray.y < 0.0)
 {yNextPlane= (yRayNotZero ?
   (octree->voxel.min.y-start.y)*yInvRay : INFINITY);
 }
 else
 {yNextPlane= (yRayNotZero ?
   (octree->voxel.max.y-start.y)*yInvRay : INFINITY);
 }
 if( intersection->ray.z < 0.0)
 {zNextPlane= (zRayNotZero ?
   (octree->voxel.min.z-start.z)*zInvRay : INFINITY);
 }
 else
 {zNextPlane= (zRayNotZero ?
   (octree->voxel.max.z-start.z)*zInvRay : INFINITY);
 }
 while( octree != (OCTREE *)NULL)
 {/* find intersections */
  content= octree->content;
  while( content != (CONTENT *)NULL)
  {help= content->object;
   if( (help->identification != identification) &&
       (help->IntersectionWithObject !=
       (INTERSECTIONOBJECTPTR)NULL))
   {if( (*help->IntersectionWithObject)( help,
          intersection))
    {location.x= intersection->start.x +
              lambda*intersection->ray.x;
     location.y= intersection->start.y +
              lambda*intersection->ray.y;
     location.z= intersection->start.z +
              lambda*intersection->ray.z;
     location.w= 1.0;
     intersect= TRUE;
    }
   help->identification= identification;
  }
  content= content->next;
 }
 if( intersect && PointInVoxel( &location,
                              &octree->voxel))
 {/* found the nearest intersection */
  octree= (OCTREE *)NULL;
 }
 else
 {/* Strahlengenerator */
  if( (xNextPlane < yNextPlane) &&
      (xNextPlane < zNextPlane))
  {currentDirectionMask= LEFTRIGHT;
   yNextPlane -= xNextPlane;
   zNextPlane -= xNextPlane;
   xNextPlane= xDistance[depth];
  }
  else
  {if( yNextPlane < zNextPlane)
   {currentDirectionMask= DOWNUP;
    xNextPlane -= yNextPlane;
    zNextPlane -= yNextPlane;
    yNextPlane= yDistance[depth];
   }
   else
   {currentDirectionMask= BACKFRONT;
    xNextPlane -= zNextPlane;
    yNextPlane -= zNextPlane;
    zNextPlane= zDistance[depth];
  }}
  oldDepth= depth;    /* go up */
  do
  {if( (currentDirectionMask == LEFTRIGHT) ||
     ((octree->direction & LEFTRIGHT) !=
     (mainDirection & LEFTRIGHT)))
   {xNextPlane += xDistance[depth];
   }
   if( (currentDirectionMask == DOWNUP) ||
    ((octree->direction & DOWNUP) !=
    (mainDirection & DOWNUP)))
   {yNextPlane += yDistance[depth];
   }
   if( (currentDirectionMask == BACKFRONT) ||
    ((octree->direction & BACKFRONT) !=
    (mainDirection & BACKFRONT)))
   {zNextPlane += zDistance[depth];
   }
   found= ((octree->direction != NODIRECTION) ?
    ((octree->direction & currentDirectionMask) !=
    (mainDirection & currentDirectionMask)) : FALSE);
   directions[depth--]= octree->direction;
   octree= octree->parent;
  }
  while( (octree != (OCTREE *)NULL) && !found);
  if( found)
  {/* go down */
   while( octree->hasChildren && (depth < oldDepth))
   {depth++;
    octree=octree->
        child[directions[depth]^currentDirectionMask];
         if( (currentDirectionMask == LEFTRIGHT) ||
     ((octree->direction & LEFTRIGHT) !=
      (mainDirection & LEFTRIGHT)))
    {xNextPlane -= xDistance[depth];
    }
    if( (currentDirectionMask == DOWNUP) ||
     ((octree->direction & DOWNUP) !=
     (mainDirection & DOWNUP)))
    {yNextPlane -= yDistance[depth];
    }
    if( (currentDirectionMask == BACKFRONT) ||
     ((octree->direction & BACKFRONT) !=
     (mainDirection & BACKFRONT)))
    {zNextPlane -= zDistance[depth];
   }}
   if( depth == oldDepth)
   {/* smaller neighbour ? */
    while( octree->hasChildren)
    {depth++;
     direction= 0;
     if( (currentDirectionMask == LEFTRIGHT) ||
      (xNextPlane > xDistance[depth]))
     {direction |=
       ((mainDirection & LEFTRIGHT) ^ LEFTRIGHT);
     }
     else
     {direction |= (mainDirection & LEFTRIGHT);
     }
     if( (currentDirectionMask == DOWNUP) ||
      (yNextPlane > yDistance[depth]))
     {direction |=
       ((mainDirection & DOWNUP) ^ DOWNUP);
     }
     else
     {direction |= (mainDirection & DOWNUP);
     }
     if( (currentDirectionMask == BACKFRONT) ||
       (zNextPlane > zDistance[depth]))
     {direction |=
       ((mainDirection & BACKFRONT) ^ BACKFRONT);
     }
     else
     {direction |= (mainDirection & BACKFRONT);
     }
     octree= octree->child[direction];
     if( (currentDirectionMask == LEFTRIGHT) ||
      ((octree->direction & LEFTRIGHT) !=
      (mainDirection & LEFTRIGHT)))
     {xNextPlane -= xDistance[depth];
     }
     if( (currentDirectionMask == DOWNUP) ||
      ((octree->direction & DOWNUP) !=
      (mainDirection & DOWNUP)))
     {yNextPlane -= yDistance[depth];
     }
     if( (currentDirectionMask == BACKFRONT) ||
      ((octree->direction & BACKFRONT) !=
      (mainDirection & BACKFRONT)))
     {zNextPlane -= zDistance[depth];
 }}}}}}
 return( intersect);
}
GLOBAL BOOLEAN InitOctree( OBJECT *header, CONST WORD
maxDepth)
{WORD depth;
 VOXEL voxel;
 maxOctreeDepth= MIN( maxDepth, MAXOCTREEDEPTH);
 voxel.min= maxVector;
 voxel.max= minVector;
 MinimizeMaximizeObject( header, &voxel);
 if( (octreeRoot= AllocateOctree( (OCTREE *)NULL,
NODIRECTION, &voxel)) !=
  (OCTREE *)NULL)
 {octreeRoot->content= AllocateContentList( header);
  DivideOctree( octreeRoot, 0);
  /* precalculating */
  xLength[0]= voxel.max.x-voxel.min.x;
  yLength[0]= voxel.max.y-voxel.min.y;
  zLength[0]= voxel.max.z-voxel.min.z;
  for( depth= 1; depth <= maxOctreeDepth; depth++)
  {
   xLength[depth]= 0.5*xLength[depth-1];
   yLength[depth]= 0.5*yLength[depth-1];
   zLength[depth]= 0.5*zLength[depth-1];
  }
  identification= (LONGWORD)0;
 }
 return( TRUE);
}
GLOBAL BOOLEAN TermOctree()
{FreeOctree( octreeRoot);
 return( TRUE);
}        ;         © 1993 M&T
```

»octree.c«: Der Octree-Algorithmus, der erste Schritt zum Raytracer, die nötigen Includes etc. finden Sie auf den folgenden Seiten – alle Listings auch auf der PD-Diskette zum Heft (➽ Seite. 114)

rechnet werden kann, da sie für alle Tiefen im Baum jeweils konstant sind. Die letzteren ändern sich »nur« additiv bzw. subtraktiv, es sind *keine* Multiplikationen bzw. Divisionen nötig. Dabei wird geschickt die Position der Zelle in der übergeordneten Zelle ausgenutzt. Denn liegt die Kinder-Zelle in derselben Richtung in der übergeordneten Zelle, in der auch der Sehstrahl läuft und befindet sich der nächste Nachbar in einer der beiden anderen Richtungen, z.B. beide nach oben und Wechsel nach rechts, so muß keine Anpassung vorgenommen werden (in der Skizze Beispiel A), da der Abstand zur jeweiligen nächsten Ebene (in diesem Fall d) sich nicht verändert hat.

Strahlenverfolgung von Kind zu Kind

Liegt die Kinder-Zelle dagegen in der entgegengesetzten Richtung oder wird die Richtung bearbeitet, in der die Zelle gewechselt wird, z.B. ist die Kinder-Zelle ein unteres Kind, der Sehstrahl läuft aber nach oben oder aber es wird nach rechts gewechselt und der Abstand nach rechts wird angepaßt (in der Skizze jeweils Beispiel B), muß der Abstand zu der betreffenden nächsten Ebene (in diesem Fall d bzw. c) um den Abstand der Ebenen untereinander (von e zu d bzw. von b zu c) erhöht werden, da dann ja gewissermaßen noch eine volle Zelle mit durchlaufen werden muß. Beim Absteigen läuft der Vorgang genauso ab, nur daß jetzt die Größen nicht durch Addition vergrößert, sondern durch Subtraktion verkleinert werden. Wurde eine gleich große, ohne Kinder-Zelle, bzw. eine größere Zelle gefunden, kann direkt weitergemacht werden, da alle Größen automatisch beim Auf- und Abstieg an die Größe der neuen Zelle angepaßt wurden.

Wurde eine gleich große mit Kinder-Zellen gefunden, wird *ohne* Konstruktion eines Punkts direkt aus »xDistance«, »yDistance« und »zDistance« sowie »xNextPlane«, »yNextPlane« und »zNextPlane« die passende Kinder-Zelle bestimmt und dabei wie oben diese Größen wieder angepaßt. Die jeweils passende Kinder-Zelle wird dabei folgendermaßen ermittelt:

Von den drei möglichen Entscheidungen bezüglich der Richtung (links oder rechts, unten oder oben, hinten oder vorne) ist eine von vorneherein festgelegt durch die Richtung, in der gewechselt wird. Denn findet der Wechsel nach rechts statt, kommen natürlich nur linke Kinder in Frage, da die rechten keine Berührungsfläche mit der Ausgangs-Zelle haben. Bei den beiden anderen wird an Hand der Abstände zur nächsten Ebene in dieser Richtung entschieden; ist der Abstand vom Durchtrittspunkt kleiner oder gleich dem Abstand der Ebenen in der nächsten Tiefe (in der Skizze Beispiel A mit behandelter Richtung unten/oben, Abstand vom Punkt A zur Ebene d kleiner als der Abstand von e zu d), so muß die Zelle in der Richtung gewählt werden, in der auch der Sehstrahl verläuft, andernfalls (in der Skizze Beispiel B mit behandelter Richtung unten/oben, der Abstand vom Punkt B zur Ebene d ist größer als der Abstand von e zu d) die andere. Das wird durchgeführt, bis eine Zelle ohne Kinder gefunden wird. Dann kann auch hier der nächste Nachbar direkt mit dem Verfahren nach Müller gesucht werden, da sämtliche Größen automatisch beim Absteigen angepaßt wurden.

```
/* File mathdefs.h */
#define EPSILON  ((FLOATING)1E-6)      /* epsilon */
#define INFINITY ((FLOATING)1E99)      /* infinity */
#define MIN(x,y)  (((x)<(y))?(x):(y))
#define MAX(x,y)  (((x)>(y))?(x):(y))
typedef struct tvector          /* vector */
  {FLOATING x, y, z, w;   } VECTOR;
GLOBAL VECTOR minVector;
GLOBAL VECTOR maxVector;
GLOBAL FLOATING Abs( CONST FLOATING);
GLOBAL VOID MinimizeVector( VECTOR *,
        CONST VECTOR *, CONST VECTOR *);
GLOBAL VOID MaximizeVector( VECTOR *,
        CONST VECTOR *, CONST VECTOR *); © 1993 M&T
```

»mathdefs.h«: Include-Datei mit mathematischen Funktionen

```
/* File octree.h */
GLOBAL BOOLEAN IntersectionWithOctree( OBJECT *,
                        INTERSECTION *);
GLOBAL BOOLEAN InitOctree( OBJECT *, CONST WORD);
GLOBAL BOOLEAN TermOctree();           © 1993 M&T
```

»octree.h«: Die Include-Datei zum Octree-Algorithmus

Die Sache mit den Voxels

Man kann allerdings mit diesem Verfahren nur direkt beginnen, falls der Startpunkt des Sehstrahls in dem die Objekte umschließenden Voxel liegt. Andernfalls muß der nächstgelegene Schnittpunkt des Sehstrahls mit diesem Voxel gefunden und von dort dann mit dem Algorithmus begonnen werden.

```
/* File intersection.h */
typedef struct tintersection        /* intersection */
{VECTOR  start, ray;     /* start and ray */
 FLOATING lambda;     /* start+lambda*ray */
 /* some additional information */
} INTERSECTION;        © 1993 M&T
```

»intersection.h«: Für die Berechnung der Schnittpunkte

```
/* File voxel.h */
typedef struct tvoxel          /* voxel */
{VECTOR min, max;
} VOXEL;
GLOBAL VOID MinimizeMaximizeVoxel( VOXEL *,
       CONST VECTOR *);
GLOBAL VOID UnionVoxel( VOXEL *, CONST VOXEL *,
       CONST VOXEL *);
GLOBAL BOOLEAN PointInVoxel( CONST VECTOR *,
       CONST VOXEL *);
GLOBAL BOOLEAN RayVoxelIntersection( FLOATING *,
    CONST VECTOR *, CONST VECTOR *, CONST VOXEL *);
                 © 1993 M&T
```

»voxel.h«: Behandlung von Voxel, d.h vereinfachten Objekten

```
/* File typedefs.h */
#include <math.h>
/* Define for memory use */
extern void *malloc( unsigned int);
extern void free( void *);
/* Other definitions */
#define CONST   const     /* constant */
#define STATIC  static    /* static */

typedef void   VOID;          /* void */
typedef char   CHAR;          /* for characters only */
typedef signed char  BYTE; /* signed 8 bit */
typedef unsigned char UBYTE;   /* unsigned 8 bit */
typedef short   WORD;          /* signed 16 bit */
typedef unsigned short UWORD; /* unsigned 16 bit */
typedef long   LONGWORD;       /* signed 32 bit */
typedef unsigned long ULONGWORD; /*unsigned 32 bit*/
typedef double  FLOATING;      /* floating point */
typedef short   BOOLEAN;       /* boolean */
#define FALSE  ((BOOLEAN)0)    /* false */
#define TRUE   (!FALSE)        /* true */
typedef unsigned int SIZE;     /* size */
#define NULLSIZE  ((SIZE)0L)   /* no size */
#define Allocate(t)     ((t *)malloc(sizeof(t)))
#define AllocateArray(t,c)  ((t *)malloc((SIZE)
                      (c)*sizeof(t)))
#define Free(p)  free((VOID *)p)        © 1993 M&T
```

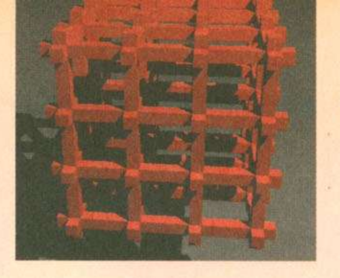
»typedefs.h«: Hier definieren wir wichtige Datentypen

Bevor man das Verfahren auf die Sehstrahlen losläßt, ist der betreffende Baum zu erzeugen. Das läßt sich einfach durch Rekursion lösen. Zuerst wird eine Zelle erzeugt, welche gerade so groß ist, daß sie alle Objekte umfaßt. Dann wird solange geteilt, bis eine maximale Tiefe erreicht ist oder aber gar kein oder nur noch ein Objekt dieser zu bearbeitenden Zelle zugeordnet sind. Beim Teilen werden acht Kinder-Zellen erzeugt und die Objekte dieser Zelle gegen die Kinder-Zellen geschnitten.

Bitte lesen Sie weiter auf Seite 112
Seite 111 Listing (Voxel.c).

```
/* File object.h */
#define OBJECT2D     1       /* types */
#define OBJECT3D    10
#define PLAIN        1       /* subtypes */
#define TRIANGLE     2
#define RECTANGLE    3
#define DISC         4
#define POLYGON      5
#define SPHERE      10
#define ELLIPSOID   20
#define CUBOID   30
#define QUADRIC   40
#define TORUS   50
typedef struct tobject          /* object */
{struct tobject *next;          /* next object */
 WORD   type, subtype;          /* type */
 VOID    *description;          /* description */
 VOXEL   voxel;                 /* surrounding voxel */
 BOOLEAN unlimited;             /* object unlimited ? */
 BOOLEAN (*VoxelIntersectedByObject)
        ( CONST struct tobject *, CONST VOXEL *);
 BOOLEAN    (*IntersectionWithObject)
          ( struct tobject *, INTERSECTION *);
 /* some additional information */
 ULONGWORD  identification;     /* identification */
} OBJECT;
typedef BOOLEAN (*VOXELINTERSECTEDPTR)
        ( CONST OBJECT *, CONST VOXEL *);
typedef BOOLEAN (*INTERSECTIONOBJECTPTR)
       ( OBJECT *, INTERSECTION *);      © 1993 M&T
```

»object.h«: Definition der einzelnen Objekte (14)



```
/* File voxel.c */
#include "typedefs.h"
#include "import.h"
#include "mathdefs.h"
#include "export.h"
#include "voxel.h"
GLOBAL VOID MinimizeMaximizeVoxel( VOXEL *voxel, CONST VECTOR *vector)
{MinimizeVector( &voxel->min, &voxel->min, vector);
 MaximizeVector( &voxel->max, &voxel->max, vector);
}
GLOBAL VOID UnionVoxel( VOXEL *result, CONST VOXEL *voxel1,
                        CONST VOXEL *voxel2)
{MinimizeVector( &result->min, &voxel1->min, &voxel2->min);
 MaximizeVector( &result->max, &voxel1->max, &voxel2->max);
}
GLOBAL BOOLEAN PointInVoxel( CONST VECTOR *point, CONST VOXEL *voxel)
{return( ((voxel->min.x <= point->x) && (point->x <= voxel->max.x)) &&
         ((voxel->min.y <= point->y) && (point->y <= voxel->max.y)) &&
         ((voxel->min.z <= point->z) &&  (point->z <= voxel->max.z)));
}
GLOBAL BOOLEAN RayVoxelIntersection( FLOATING *lambda,
               CONST VECTOR *start,
               CONST VECTOR *ray, CONST VOXEL *voxel)
{FLOATING nearest, farest, lambda1, lambda2, help;
 nearest= -INFINITY;
 farest= INFINITY;
 if( Abs( ray->x) > EPSILON)
 {lambda1= (voxel->min.x-start->x)/ray->x;
  lambda2= (voxel->max.x-start->x)/ray->x;
  if( lambda1 > lambda2)
  {help= lambda1;
   lambda1= lambda2;
   lambda2= help;   }
  if( lambda1 > nearest) {nearest= lambda1;   }
  if( lambda2 < farest)  {farest= lambda2;    }
  if( (nearest > farest) || (farest < 0.0))
  {return( FALSE);
 }}
 else
 {if( (start->x < voxel->min.x) || (start->x > voxel->max.x))
  {return( FALSE);
 }}
 if( Abs( ray->y) > EPSILON)
 {lambda1= (voxel->min.y-start->y)/ray->y;
  lambda2= (voxel->max.y-start->y)/ray->y;
  if( lambda1 > lambda2)
  {help= lambda1;
   lambda1= lambda2;
   lambda2= help;
  }
  if( lambda1 > nearest) {nearest= lambda1;   }
  if( lambda2 < farest)  {farest= lambda2;    }
  if( (nearest > farest) || (farest < 0.0))
  {return( FALSE);
 }}
 else
 {if( (start->y < voxel->min.y) ||
 (start->y > voxel->max.y))
  {return( FALSE);
 }}
 if( Abs( ray->z) > EPSILON)
 {lambda1= (voxel->min.z-start->z)/ray->z;
  lambda2= (voxel->max.z-start->z)/ray->z;
  if( lambda1 > lambda2)
  {help= lambda1;
   lambda1= lambda2;
   lambda2= help;
  }
  if( lambda1 > nearest)  {nearest= lambda1;  }
  if( lambda2 < farest)   {farest= lambda2;   }
  if( (nearest > farest) || (farest < 0.0))
  {return( FALSE);
 }}
 else
 {if( (start->z < voxel->min.z) ||
 (start->z > voxel->max.z))
  {return( FALSE);
 }}
*lambda= nearest;
return( TRUE);
}                                   © 1993 M&T
```

»voxel.c«: Wichtig für das Octree-Verfahren: Die Implementation der Routinen zur Behandlung der sog. Voxel

```
/* File mathdefs.c */
#include "typedefs.h"
#include "import.h"
#include "export.h"
#include "mathdefs.h"
GLOBAL VECTOR minVector =
       { -INFINITY, -INFINITY, -INFINITY, 1.0};
GLOBAL VECTOR maxVector =
       { INFINITY, INFINITY, INFINITY, 1.0};
GLOBAL FLOATING Abs( CONST FLOATING number)
{return( (number < 0.0) ? -number : number);
}
GLOBAL VOID MinimizeVector( VECTOR *result,
                            CONST VECTOR *vector1,
                            CONST VECTOR *vector2)
{result->x= MIN( vector1->x, vector2->x);
 result->y= MIN( vector1->y, vector2->y);
 result->z= MIN( vector1->z, vector2->z);
 result->w= 1.0;
}
GLOBAL VOID MaximizeVector( VECTOR *result,
                            CONST VECTOR *vector1,
                            CONST VECTOR *vector2)
{result->x= MAX( vector1->x, vector2->x);
 result->y= MAX( vector1->y, vector2->y);
 result->z= MAX( vector1->z, vector2->z);
 result->w= 1.0;
}                                    © 1993 M&T
```

»mathdefs.c«: Implementation der mathematischen Funktionen

Ist die Schnittmenge nicht leer, d.h. besitzt die Oberfläche des Objekts Punkte in dieser Zelle, wird dieses Objekt dieser neuen Zelle zugeordnet. Sind alle Objekt einer Zelle abgearbeitet, so kann mit den acht erzeugten Kinder-Zellen genauso verfahren werden.

Verbesserungen sind noch drin

Allerdings läßt sich dieses Verfahren auch noch verbessern. So tauchen ja viele Objekte mehrere Male in den vom Strahl durchlaufenen Zellen auf. Normalerweise würde jedesmal wieder erfolglos ein Schnittpunkt berechnet werden (denn wäre er erfolgreich, wäre er schon bei der Bearbeitung der Zellen weiter vorne gefunden worden). Das ist unnötig. Um das zu vermeiden, erhält die Objekt-Struktur noch eine Variable »identification«, die bei Initialisierung des Octree-Verfahrens auf Null gesetzt wird. Bei der Berechnung wird dann eine globale Variable ebenfalls mit Namen »identification« bei jedem Aufruf der Prozedur, die die Schnittpunkte berechnet, um Eins erhöht. Wenn ein Objekt geschnitten wurde, wird seiner »identification« als Markierung dafür der Wert der aktuelle globalen »identification« zugewiesen. Wenn das Objekt bei demselben Strahl nochmals in der Liste einer Zelle auftauchen sollte, wird es mit dieser »identification« bemerkt (mit der globalen identisch) und das Objekt nicht nochmals geschnitten.

Des weiteren machen natürlich Ebenen und andere unbegrenzte Objekte einige Probleme. Da ihre Ausdehnung nicht begrenzt ist, läßt sich natürlich auch kein umschließendes Voxel finden. Bei dem vorgestellten Verfahren wurde dieses Problem folgendermaßen gelöst: diese Objekte werden bei der Ermittlung der die Szene umgebenden Zelle nicht beachtet und

```
/* File import.h */

#define IMPORT          /* now importing */
#define GLOBAL extern          © 1993 M&T
```

»import.h«: Die Datei ist fürs Importieren erforderlich

```
/* File export.h */

#ifdef IMPORT
#undef IMPORT
#undef GLOBAL
#endif

#define EXPORT          /* now exporting */
#define GLOBAL
#define LOCAL static           © 1993 M&T
```

»export.h«: Fürs Exportieren erforderliche Include-Datei

die dem Objekt zugeordnete Prozedur, die bestimmt, ob dieses Objekt eine gegebenen Zelle schneidet, liefert immer den Wert Falsch, so daß diese Objekte in keiner der Zellen enthalten sind. Beim Aufruf der Routine »IntersectionWithOctree«, die die Schnittpunkte mit den Objekten in dem Octree finden soll, werden die unbegrenzten Objekte dann getrennt »vor« dem eigentlichen Algorithmus geschnitten.

Letzteres hat den Grund, da für Objekte wie Ebenen usw. ein Schnittpunkt leicht und schnell gefunden werden kann und somit eine gewisse Sortierung der anderen Objekte vorgenommen wird (nämlich »vor« und »nach« dem nächsten unbegrenzten Objekt). Falls der Sehstrahl den die Szene umgebenden Voxel verfehlt, kann natürlich einfach zurückgesprungen werden, da dann kein Schnittpunkt mit den Objekten in diesem Voxel vorliegen kann.

Diese Behandlung der unbegrenzten Objekte mag etwas umständlich erscheinen. Doch ist es für solche Objekte meist gar nicht so einfach zu entscheiden, ob ihre Oberfläche Punkte in einer bestimmen Zelle hat, aber immer mit einigem (zeitlichem) Aufwand verbunden, so daß man abwägen muß zwischen den Vor- und Nachteilen, die eine exaktere Behandlung mit sich bringt.

Mit unserem Verfahren ist es möglich, Szenen fast unabhängig von der Anzahl der dargestellten Objekte mit fast konstanter Zeit zu berechnen, da sich die Anzahl der pro Sehstrahl geschnittenen Objekte trotz zunehmender Gesamtanzahl kaum verändert, nur durch das Verfahren selber entsteht ein geringer Overhead.

Natürlich sind die Prozeduren, wie sie hier vorgestellt werden, noch nicht für sich genommen lauffähig. Es müssen für jedes Grundobjekt eine Reihe von Funktionen definiert werden, die die Octree-Routine aufruft. So wird mit »VoxelIntersectedByObject« beim Objekt erfragt, ob eine nicht-leere Schnittmenge mit der übergebenen Zelle existiert. Des weiteren muß eine Funktion »IntersectionWithObject« für jedes Grundobjekt definiert sein, die einen Schnittpunkt zwischen dem Sehstrahl und sich selbst ermittelt. Dabei entscheiden die Objekte selber, ob ihr Schnittpunkt *vor* dem bisher nächsten liegt und ersetzen ihn in diesem Fall.

Von den anderen Teilen wie z.B. »mathdefs.c« sind nur die Teile abgedruckt, die direkt von dem Algorithmus verwendet werden. Für einen funktionierenden Raytracer werden natürlich noch Dinge wie Skalarprodukt und Kreuzprodukt, Transformationen usw. benötigt.

Der Weg zum Raytracer

Dabei ist darauf zu achten, daß hier sog. homogene Koordinaten verwendet werden, die neben »x«, »y« und »z« noch eine Komponente »w« beinhalten. Hat »w« den Wert Null, handelt es sich bei dem Vektor um einen Richtungsvektor, hat es den Wert Eins, hat man es mit einen Ortsvektor zu tun. Die umständlich anmutende Technik hat den Vorteil, daß »alle« Transformationen (Translation, Rotation, Skalierung) durch 4x4-Matrizen und eine Hintereinanderausführung durch eine Matrizen-Multiplikation beschrieben werden können (nähere Informationen siehe auch hier unter [1]).

Des weiteren handelt es sich bei den verschiedenen Typen wie »OBJECT« usw. nur um reduzierte Hüllen, die noch mit Leben gefüllt werden müssen. Dabei sollte das hier vorgestellte nur als Vorschlag aufgefaßt werden, der natürlich den Algorithmus als solchen nicht berührt und genauso gut auch anders gelöst werden kann. Ansonsten muß im Programm vor der eigentlichen Berechnung die Initialisierung einmal aufgerufen und danach die Routine, die normalerweise die Schnittpunkte berechnet, durch die vorgestellte ersetzt werden.

Wer schreibt ein Programm?

Zum Ausprobieren sollten schon einige Dutzend Objekte in der Szene sein, da sonst der durch den Algorithmus verursachte Overhead den Zeitgewinn wieder aufhebt. Die günstigste maximale Tiefe, die nicht nur von der Anzahl der Objekte, sondern auch von deren Anordnung abhängt, läßt sich näherungsweise durch »Log(n)« bis »Log(n)+1« abschätzen, wobei »n« die Anzahl der Objekte in der Szene und »Log« der Logarithmus zur Basis 8 ist. Als Beschleunigungsfaktor dürfen Sie im günstigsten Fall bei 64 Objekten ungefähr einen Faktor von 5, bei 512 Objekten ungefähr von 30 erwarten.

Sicher ist es ein schönes Stück Arbeit, bis Sie einen richtiges Raytracing-Programm fertig haben. Aber es kann sich lohnen: Wenn Sie ein Programm haben, schicken Sie es an die Redaktion. Besonders gute Programme werden wir im nächsten Sonderheft – gegen Honorar natürlich – beschreiben und auf der PD-Diskette zum Heft veröffentlichen. ■

Literatur:

[1] Foley; van Dam; Feiner; Hughes: Computer Graphics, Principles and Practice; Addison--Wesley 1990

[2] Glassner, Andrew S. (Hrsg.): An Introduction to Raytracing; Academic Press 1989

[3] Watt, Alan; Watt, Mark: Advanced Animation and Rendering Techniques; Adisson--Wesley 1992

[4] Endl, Robert: Raytracing mittels adaptiver Octree--Nachbarsuche; Informationsschrift der Hessischen Hochschulen zur CeBIT 1992

Permutatioinen und Anagramme

Aus Müll wird Geld

In guter alter AMIGA-Knobeltradition wollen wir Sie auch im Sonderheft »Faszination Programmieren« bis zur nächsten Ausgabe mit einer neuen Aufgabe beschäftigt wissen, wobei es einen tollen Preis zu gewinnen gibt.

von G.Steffens

Schon im AMIGA-Magazin in einer früheren Ausgabe der Knobelecke [1] haben wir feststellen können, wie schwierig für unseren Computer der Umgang mit der deutschen Sprache ist. Auf der Jagd nach Anagrammen und Palindromen mußten wir versuchen, uns im Dschungel aus Buchstaben und Wörtern zurechtzufinden.

An dieser Stelle wollen wir uns wieder ins Buchstabenchaos stürzen! Im Mittelpunkt unserer Aufgabe steht die sog. Wortkette – auch Wortleiter genannt. Diese Knobelei geht auf den »Rätselkomponisten« Lewis Caroll zurück und dürfte Denksportfreunden aus diversen Rätselbüchern oder Zeitschriften bekannt sein.

Spiel mit Worten

Es werden zwei beliebige Wörter einer Sprache vorgegeben, die durch Austauschen von jeweils nur einem Buchstaben eine Kette von sinnvollen Wörtern bilden sollen. Das Ausgangswort wird auf diesem Weg – mit möglichst wenigen Schritten – ins Zielwort überführt. Zur Veranschaulichung verwandeln wir einmal MÜLL in GELD:

MÜll -> MOLL -> SOLL ->
SOLD -> GOLD -> GELD.

Wie kann man nun solche Wortketten bei gegebenen Anfangs- und Endwort mit dem Computer ermitteln? Hierzu sollen Sie ein Programm entwerfen, das immer den kürzesten Weg findet.

Betrachten wir obige Wortkette genauer: Die Worte MÜLL und GELD unterscheiden sich an drei Positionen, es sind also mindestens drei Austauschoperationen nötig, um das Ziel zu erreichen. Es läßt sich außerdem feststellen, daß mehr als drei Umwandlungsschritte zur Lösung der Aufgabe notwendig sind: Vertauschen Sie einfach die entsprechenden Buchstaben von Start- und Zielwort. Anschließend werden die neuen Wörter im Hinblick auf ihre Zulässigkeit untersucht. Für unser Müllproblem erhalten wir folgende Liste von Wörtern:

GÜLL -> MELL -> MÜLD
MELD -> GÜLD -> GELL.

Eine Untersuchung der Liste zeigt die Unzulässigkeit der Worte. Aber halt, vielleicht ist die Sache doch nicht so einfach! Es kommt bei der Zulässigkeitskontrolle auf die geeignete Wahl des Wortschatzes an, gell. Soll das Wort GELL verworfen oder als sinnvolles Wort zugelassen werden? Sicherlich werden viele Leser aus dem süddeutschen Sprachraum das Wort als »sinnvoll« empfinden, andere Leser werden hingegen nur mit dem Kopf schütteln.



»Computer-Koffer«: Preis für den besten Tüftler: ein Koffer im Computer-Design: innen Leder, außen Computerplatinen, gestiftet von v&r design, products GmbH

Damit der Amiga die Aufgabe in Angriff nehmen kann, benötigt er zur Orientierung ein Wörterbuch. Der Autor war so leichtsinnig und hat eine Liste von Wörtern im ASCII-Format zusammengestellt, wir beschränken uns dabei auf Wörter der Länge 4. Die komplette Liste finden Sie auf unserer PD-Diskette zum Heft (s. Seite 114). Die Liste soll unseren zulässigen Wortschatz darstellen. Sie sollte eine brauchbare Grundlage für Ihre Computerexperimente bilden. Selbstverständlich darf die Liste entsprechend dem persönlichen Geschmack erweitert oder verändert werden.

Sie finden auf der PD-Diskette auch ein BASIC-Programm, das sich mit einem Teilaspekt des gestellten Problems beschäftigt und Ihnen die Sache vereinfachen soll. Das Programm sucht zu einem Wort alle Wörter aus der Liste, die sich vom Ausgangswort an genau einer Stelle unterscheiden. Es wird sich bei Ihren Bemühungen sicher als nützlich erweisen.

Hier nochmals die Aufgabe: Wer uns das interessanteste – dokumentierte! – Programm schickt, das zu zwei gegebenen Worten immer eine Querverbindung findet – basierend auf einem festen Wortschatz in Form eines Wörterbuchs (ASCII-Datei) und möglichst schnell – gewinnt, neben dem üblichen Honorar für die Veröffentlichung, einen tollen Computerkoffer, gestiftet von v&r design. Also, Mitmachen lohnt sich. Viel Spaß und Glück. ■

Adresse: v&r design products GmbH, Platanenstr. 6, 40233 Düsseldorf, Tel: 02 11/67 68 43; Fax: 02 11/67 69 46

Literartur:

[1] Steffens, Gerald: Knobelecke "Wortspiele", AMIGA-Magazin 11/92
[2] Uphoff, Matthias: "Das Software-Experiment", PC Schneider International 1/87 + 4/87
[3] Dewdney, Alexander Keewatin: "Verwandlungskünste mit Worten", Computerkurzweil III (Sonderheft 8), Spektrum der Wissenschaft



Faszination Programmieren Nr. 3

Noch´n Gedicht

Die nächste Ausgabe von »Faszination Programmieren« ist bereits geplant. Anfang nächsten Jahres wird es soweit sein. Als Schwerpunkte sind geplant:

- Alles rund um OS 3.0;
- Der 68030er und seine Geheimnisse;
- Programmierung der neuen Amigas;
- Programmierkurse in C und Assembler.

Und natürlich wieder Tips & Tricks, tolle Tools und Knobeleien in Hülle und Fülle.

Wenn Sie sich an der nächsten Ausgabe als Autor beteiligen möchten, schicken Sie Ihre Vorschläge an die Redaktion. Aber fackeln Sie nicht zu lange – die Zeit läuft.

Schreiben Sie uns auch, wie Ihnen diese Ausgabe gefallen hat? Sind Sie mit der Mischung zufrieden, oder wollen Sie mehr zu Assembler, C oder anderen Sprache lesen? Wie wär´s mit ARexx, C++ oder Oberon? Was wünschen Sie sich für die nächsten Ausgaben am meisten: Kurse, Tips & Tricks, Grundlagen, Knobelaufgaben oder, oder, oder?

Unsere Adresse:
AMIGA-Redaktion
»Faszination Programmieren«
Markt & Technik Verlag
Hans-Pinsel-Str. 2
85540 Haar bei München

Nr.1 gibts noch!

Für alle, die unsere Erstausgabe verpaßt haben, die gute Nachricht: Nummer 1 des Sonderhefts »Faszination Programmieren« kann jetzt wieder nachbestellt werden und zwar bei folgender Adresse:

PVS Fulfillment
74172 Neckarsulm
Tel.: 0 71 32/9 69-181
Fax: 0 71 32/9 69-190

PD-Disketten zum Heft

Muß man haben

Unverbindliche Preisempfehlung: 3,90 Mark



Wie bei unserer Nummer 1 des Sonderhefts »Faszination Programmieren« gibts auch diesmal alle Listings und viele Extras auf PD-Disketten.

Public Domain ist schon eine feine Sache: Für rund 5 Mark bekommt man eine Diskette voll mit Programmen, Tools etc... Und das Ganze darf kopiert werden und steht so jedem Amiga-Besitzer kostengünstig zur Verfügung.

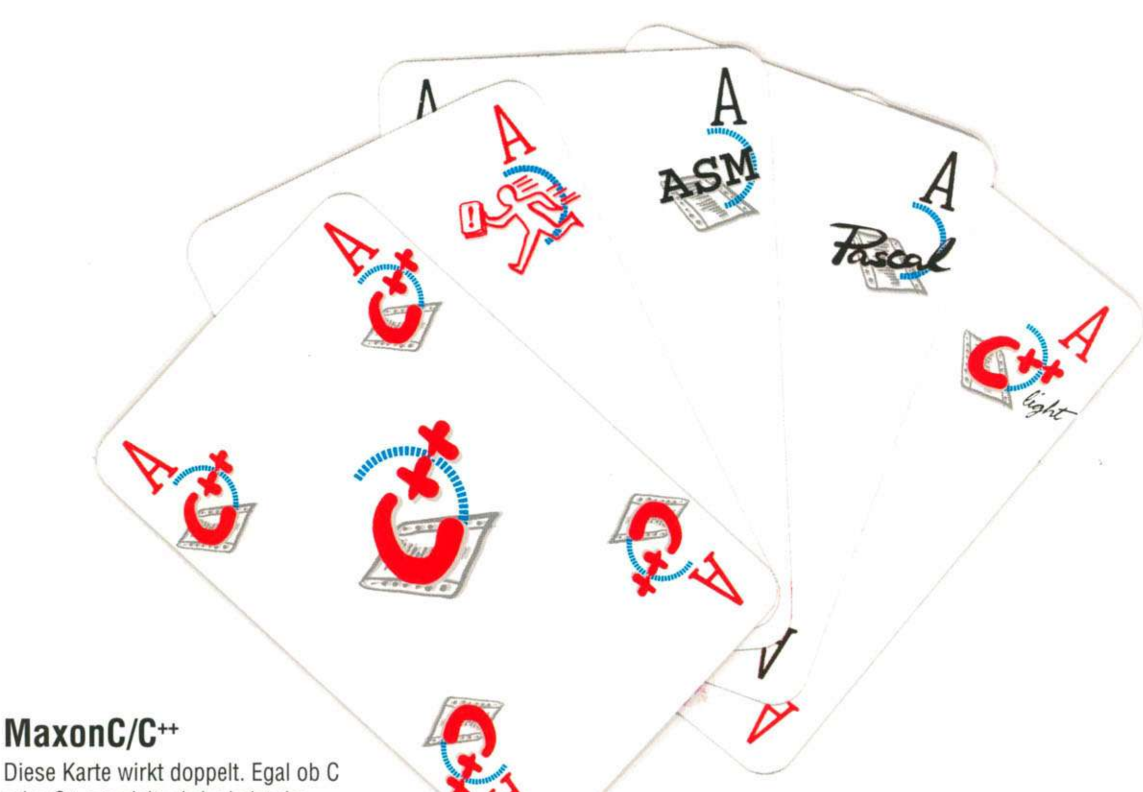
Und da wir wollen, daß auch alle Programme aus diesem Sonderheft jedem Amiga-Besitzer zu Gute kommen, gibts alle Listings und Programme aus diesem Heft als Public Domain verteilt auf mehrere Disketten:

- Disk 1 enthält die in diesem Heft abgedruckten Listings sowie – im Falle von Compilersprachen und Assembler – die lauffähigen Programme . Zusätzlich finden Sie im Heft erwähnte Listings und Programme, die für einen Abdruck zu lang waren, und ein kurzes Wörterbuch im ASCII-Format, mit dem Sie die Knobelaufgabe von Seite 113 lösen können.
- Auf Disk 2 finden Sie den Installer von Commodore sowie zahlreiche im ARexx-Schwerpunkt ab Seite 84 erwähnte Libraries.
- Auf Disk 3 bieten wir Ihnen eine lauffähige Oberon-Demo mit der Sie die Oberon-Listings aus diesem Sonderheft übersetzen können.
- Disk 4 und 5 enthalten eine C++-Demo.

Sie können natürlich auch noch die Disketten unserer Erstausgabe nachbestellen. Oder Sie besorgen sich die Disketten bei Ihrem PD-Händler oder über Compuserve (go mut).

Programmieren erfordert Werkzeuge, auf die man sich verlassen kann. MAXON ist seit Jahren der renommierteste deutsche Hersteller von Programmiersprachen und -Tools für den AMIGA. Dabei wird größter Wert auf Komfort, Schnelligkeit und Zuverlässigkeit gelegt. Das spiegelt sich in allen unseren Sprachen wieder. Sie garantieren schnelle Einarbeitung und effiziente Programmentwicklung.
Mit Programmiersprachen von MAXON haben Sie immer gute Karten.

DAS TRAUMBLATT ALLER PROGRAMMIERER



MaxonC/C++

Diese Karte wirkt doppelt. Egal ob C oder C++ gespielt wird; sie ist der höchste Trumpf und läßt sich bei allen Gelegenheiten einsetzen.

MaxonC/C++ light

Zweithöchste Karte im Spiel, nur die obige zählt mehr. Ideal zum Austesten und trotzdem stärker als alle anderen C-Karten.

HotHelp

Die Spezialkarte. Wirkt immer dann, wenn man nicht so recht weiß, was man spielen soll.

KickPascal

Der höchste Trumpf beim Pascal-Spiel. Ideal für alle Liebhaber des geordneten Spiels.

MaxonASM

Assembler ist das schwierigste Spiel. Wenn man diese Karte spielt, kann man jedoch locker gewinnen.

Maxon C/C++ V 1.1
Kombiniertes C und C++ Entwicklungssystem. Komplett mit Editor, Compiler, Oberflächengenerator und HotHelp DM 398.-*

Maxon C/C++ developer
Die Profiversion. Zusätzlich mit Source-Level Debugger und ext. Assembler DM 598.-*

Maxon C/C++ light
Einstiegsversion mit Editor und uneingeschränktem Compiler DM 149.-*

MaxonASM
680x0-Assembler mit Editor, Reassembler, Monitor und Debugger DM 149.-*

KickPascal
Schneller Pascal-Compiler, sowohl für Profis als auch für Anfänger. DM 249.-*

HotHelp
Online Help-System mit der kompletten OS 2.1-Dokumentation DM 89.-*

Reizen Sie Ihren AMIGA aus!



MAXON Computer GmbH • Industriestr. 26 • 65734 Eschborn • Tel.: 06196 / 481811 • Fax: 06196 / 41885

* alle Preise sind unverbindlich empfohlene Verkaufspreise

DISK EXPANDER

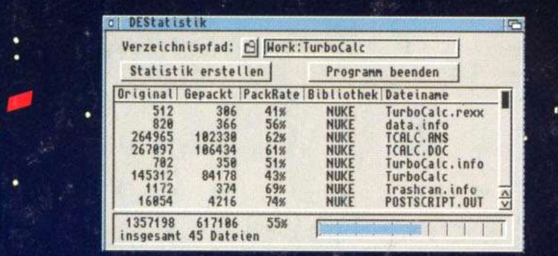
KOMPRESSIONS-SOFTWARE DER SPITZENKLASSE

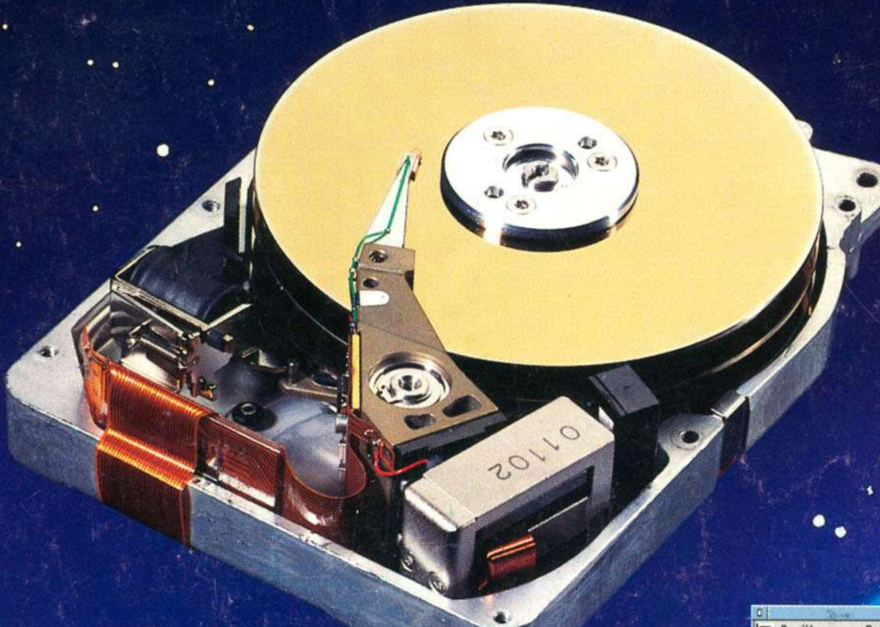
DiskExpander ist die Top-Neuheit für alle Amiga-User. Mit DiskExpander können Sie die Kapazität Ihrer Festplatte und Ihrer Diskettenlaufwerke etwa verdoppeln. Die Installation erfolgt in Sekundenschnelle und anschließend arbeitet der DiskExpander unsichtbar im Hintergrund.
Die Daten werden auf ca. 30 bis 70% der ursprünglichen Größe reduziert und verschiedene Kompressionsalgorithmen stehen zur Wahl.
Das geniale Programmkonzept sorgt dafür, daß auch Einsteiger den DiskExpander auf Anhieb optimal einsetzen können. Zur Installation, die weitgehend automatisiert erfolgt, benötigen Sie keinerlei Spezialkenntnisse. Der DiskExpander erhöht nicht nur die Kapazität Ihrer Festplatte. Auch auf normalen Disketten können Sie im Durchschnitt ab sofort ca. 1,5MB Software unterbringen, und selbst die RAD-Disk können Sie problemlos verdoppeln.
Kennen Sie eine bessere Möglichkeit, Ihren Amiga für wenig Geld sinnvoll zu erweitern?

Bei der Entwicklung von DiskExpander wurde größtmöglicher Wert auf Datensicherheit, variable Kompression und gute Geschwindigkeit gelegt. Sie können selbst bestimmen, ob Sie Ihre Daten hochgradig verdichten wollen, oder ob Ihnen mittlere Kompressionsraten ausreichen und haben somit direkten Einfluß auf die Geschwindigkeit. Selbstverständlich können Sie auch Ihren bevorzugten xpk-Packer einsetzen.
Insgesamt sieben Programmierer aus fünf verschiedenen Nationen haben entscheidend dazu beigetragen, diese technische Meisterleistung zu entwickeln.
DiskExpander wurde über einen Zeitraum von mehreren Monaten weltweit von mehr als 100 Benutzern getestet und für gut befunden!
DiskExpander wird mit deutscher Benutzeroberfläche ausgeliefert, kann auf drei verschiedene Arten installiert werden und wird auch in Zukunft ständig weiterentwickelt.

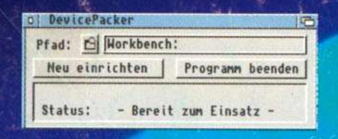
- **Warnung!** Es wird dringend davor gewarnt,
- illegale Kopien von DiskExpander zu benut-
- zen, da diese in der Regel modifiziert wur-
- den und die Sicherheit Ihrer Daten in keiner
- Weise gewährleisten!

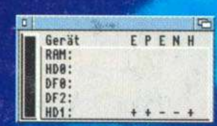
DM 69,-













DiskExpander SV
DiskExpander SV ist eine spezielle Unpack-Only-Version, die wir jedem Softwareproduzenten in einem preiswerten Lizensierungsverfahren anbieten. So können Sie und Ihre Kunden von der innovativen DiskExpander-Technologie profitieren. Erfragen Sie gleichzeitig unsere vorteilhaften Bundling-Preise.

Attention
The international version of DiskExpander will be available soon and we are looking for additional distributors. Please send your requests by FAX and we will reply immediatly including further details.

Versandkosten
Inland: DM 4,- V-Scheck
DM 8,- Nachnahme
Ausland: DM 8,- V-Scheck
DM 25,- Nachnahme

Stefan Ossowski´s Schatztruhe
Gesellschaft für Software mbH
Veronikastraße 33
45131 Esssen
Telefon 02 01/78 87 78
Telefax 02 01/79 84 47

Der Disk Expander läuft auf allen Commodore Amiga 500, 600, 1000, 1200, 2000, 2500, 3000 (T) und 4000 (T) unter Kickstart 1.2, 1.3, 2.0 und 3.0 mit oder ohne installierter Festplatte.